Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____Комп'ютерних наук_____
(повна назва)

Кафедра _____Інформаційних управляючих систем_____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _____другий (магістерський)_____

Дослідження і класифікація інструментальних засобів розробки front-end-застосунків як ресурсів IT-проєкту
(тема)

Виконав:
студент 2 курсу, групи УПГІТм-21-1
_____Дмитро КІНДІЙ_____
(власне ім'я, прізвище)

Спеціальність ____122 Комп'ютерні науки_____
(код і повна назва спеціальності)
Тип програми ____освітньо-наукова_____
(освітньо-професійна або освітньо-наукова)
Освітня програма Управління проєктами в галузі інформаційних технологій
(повна назва освітньої програми)

Керівник проф. каф. ІУС Максим ЄВЛАНОВ
(посада, власне ім'я, прізвище)

Допускається до захисту

Зав. кафедри _____ ___Костянтин ПЕТРОВ___
(підпис) (власне ім'я, прізвище)
2023 р.

Харківський національний університет радіоелектроніки

Факультет _____Комп'ютерних наук_____

Кафедра _____Інформаційних управляючих систем_____

Рівень вищої освіти _____другий (магістерський)_____

Спеціальність _____122 Комп'ютерні науки_____
(код і повна назва)

Тип програми _____освітньо-наукова_____
(освітньо-професійна або освітньо-наукова)

Освітня програма_Управління проєктами в галузі інформаційних технологій
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«_03_»__квітня__ 20 23_ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові_____Кіндію Дмитру Дмитровичу_____
(прізвище, ім'я, по батькові)

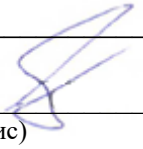
1. Тема роботи Дослідження і класифікація інструментальних засобів розробки front-end-застосунків як ресурсів IT-проєкту
затверджена наказом університету від _03___квітня__2023_ р. № _319Ст_
2. Термін подання студентом роботи до екзаменаційної комісії _18_ ____травня___ 2023 р.
3. Вихідні дані до роботи інструменти розробки frontend-застосунків, сучасні дослідження продуктивності веб-сайтів, методи SEO оптимізації веб-додатків, головні метрики продуктивності.
4. Перелік питань, що потрібно опрацювати в роботі проаналізувати інструменти frontend-розробки та методи оптимізації веб-сайту, дослідити показники продуктивності і стратегії рендерингу з виявленням переваг та недоліків, адаптувати базові стратегії рендерингу, розробити змішані стратегій під сучасні вимоги, порівняти особливості стратегій рендерингу.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз інструментів розробки frontend-застосунків	03.04.2023 – 05.04.2023	Виконано
2	Аналіз важливості швидкості та продуктивності веб-сайту	06.04.2023 – 07.04.2023	Виконано
3	Аналіз проблем та ключових факторів продуктивності сучасних веб-сайтів	08.04.2023 – 10.04.2023	Виконано
4	Огляд методів оптимізації, показників продуктивності, видів рендерингу веб-сторінок	11.04.2023 – 14.04.2023	Виконано
5	Опис постановки задачі дослідження	14.04.2023 – 15.04.2023	Виконано
6	Опис показників продуктивності і стратегій рендерингу з виявленням переваг та недоліків	16.04.2023 – 19.04.2023	Виконано
7	Адаптація базових стратегій рендерингу та розробка змішаних стратегій під сучасні вимоги	20.04.2023 – 29.04.2023	Виконано
8	Порівняння особливостей стратегій рендерингу	30.04.2023 – 31.04.2023	Виконано
9	Оформлення пояснювальної записки	01.05.2023 – 11.05.2023	Виконано
10	Оформлення графічної частини та презентаційних матеріалів захисту	12.05.2023 – 14.05.2023	Виконано
11	Представлення на рецензування	15.05.2023	Виконано
12	Представлення дипломного проекту в ЕК	19.05.2023	Виконано

Дата видачі завдання _03__ ___квітня_____2023 р.

Студент _____ _____
(підпис)

Керівник роботи _____  _____ професор каф. ІУС Максим ЄВЛАНОВ
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи містить: 76 с., 4 розділи, 18 рис., 2 табл., 13 джерел.

АНАЛІЗ, ВЕБ-ДОДАТОК, ДОСЛІДЖЕННЯ, СТРАТЕГІЇ РЕНДЕРИНГУ, МЕТРИКИ ПРОДУКТИВНОСТІ, РЕНДЕРИНГ, ФРЕЙМВОРКИ, SEO

У кваліфікаційній роботі виконано дослідження показників продуктивності веб-додатків та стратегій рендерингу як інструментальних засобів розробки веб-застосунків в рамках ІТ-проєктів. У результаті проведеного аналізу було адаптовано існуючі стратегії рендерингу під сучасні вимоги ринку та розроблено змішані стратегії рендерингу.

Об’єктом дослідження в рамках магістерської кваліфікаційної роботи є інформаційні ресурси інструментальних засобів розробки веб-додатків.

Предметом дослідження є стратегії рендерингу в розробці веб-застосунків та їх ефективність для сучасних проєктів.

Результати роботи:

– аналіз інструментів frontend-розробки і методів оптимізації веб-сайту;

– огляд методів оптимізації, показників продуктивності, видів рендерингу веб-сторінок;

– опис постановки задачі дослідження;

– опис показників продуктивності і стратегій рендерингу з виявленням переваг та недоліків;

– адаптація базових стратегій рендерингу та розробка змішаних стратегій під сучасні вимоги;

– порівняння особливостей стратегій рендерингу.

ABSTRACT

The explanatory note to the qualification work contains: 76 pages, 4 sections, 18 figures, 2 tables, 13 sources.

ANALYSIS, FRAMEWORKS, PERFORMANCE METRICS, RENDERING, RENDERING STRATEGIES, RESEARCH, SEO, WEB-APPLICATION

The study is about a research of performance indicators of web applications and rendering strategies as instrumental means of web applications development within the framework of IT projects. As a result of the analysis, existing rendering strategies were adapted to modern market requirements and mixed rendering strategies were developed.

The object of research of the master's qualification study is the information resources of tools for the development of web applications.

The subject of the research is rendering strategies in the development of web applications and their effectiveness for modern projects.

Work results:

– analysis of frontend development tools and website optimization methods;

– overview of optimization methods, performance indicators, types of web page rendering;

– a description of the research task statement;

– description of performance indicators and rendering strategies with identification of advantages and disadvantages;

– adaptation of basic rendering strategies and development of mixed strategies for modern requirements;

– a comparison of the features of rendering strategies.

ЗМІСТ

Скорочення та умовні познаки 8

Вступ 9

1 Аналіз інструментів frontend-розробки та методів оптимізації веб-сайту ... 11

1.1 Аналіз інструментів розробки frontend-застосунків 11

1.1.1 Опис призначення frontend-фреймворків 12

1.1.2 Опис ролі стратегії рендерингу 13

1.1.3 Призначення CSS-процесору 13

1.1.4 Роль збірників модулів у frontend-проєктах 14

1.2 Аналіз важливості швидкості та продуктивності веб-сайту 15

1.3 Аналіз проблем та ключових факторів продуктивності сучасних веб-сайтів 16

1.3.1 SEO 16

1.3.2 Швидкість завантаження сторінки 17

1.3.3 Швидкість візуалізації контенту й медіа сторінки 19

1.4 Огляд методів оптимізації, показників продуктивності, видів рендерингу веб-сторінок 21

1.4.1 Типи рендерингу (будування) сторінок 21

1.4.2 Показники продуктивності та інструменти їх аналізу 22

1.5 Опис постановки задачі дослідження 23

2 Опис показників продуктивності і стратегій рендерингу з виявленням переваг та недоліків 24

2.1 Largest Contentful Paint 26

2.2 First Input Delay 29

2.3 Cumulative Layout Shift 32
2.4 Server-Side Rendering 34
2.6 Client-Side Rendering 39
3 Адаптація базових стратегій рендерингу та розробка змішаних стратегій під сучасні вимоги 41
3.1 Поєднання рендерингу на сервері та клієнті 41
3.2 Потокове надсилання на клієнт сторінок побудованих на сервері 45
4 Апробація розроблених стратегій рендерингу 50
Висновки 54
Перелік джерел посилання 55
Додаток А Графічний матеріал ..... **Ошибка! Закладка не определена.**

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

CSR – Client-Side Rendering

DOM – Document Object Model

DX – Developer Experience

FCP – First Contentful Paint

FID – First Input Delay

JS – JavaScript

LCP – Largest Contentful Paint

PWA – Progressive Web Application

SEO – Search Engine Optimization

SSG – Static Site Generation

SSR – Server-Side Rendering

TTFB – Time to First Byte

TTI – Time to Interact

UX – User Experience

ВСТУП

Сьогодні ні в кого не виникає сумніву щодо важливості наявності веб-сайту бізнеса. Нещодавні тривалі та масштабні карантини через епідемію COVID-захворювання не тільки змінили звички та образ життя населення планети, а й спровокували великі зміни у веденні бізнесу і, зокрема, стали причиною масштабної цифровізації (діджиталізації).

Веб-сайт вже давно є невід'ємною складовою більшості видів бізнесу. Цілком прогнозованим є постійний зріст попиту на запровадження веб-сайтів для підтримки бізнесу. Глобальні події останніх років тільки сприяють цьому росту зі значним прискоренням.

Так, наприклад, під час пандемії споживачі різко перейшли до онлайн-застосунків, і компанії та галузі, у свою чергу, відреагували на це. Результати опитування консалтингової компанії McKinsey & Company підтверджують швидкий перехід до взаємодії з клієнтами через цифрові канали [1]. Вони також показують, що темпи цифровізації на роки випереджають показники, які були під час проведення попередніх досліджень, і навіть більше в розвинених країнах Азії, ніж в інших регіонах. Зараз принаймні 80 відсотків взаємодій з клієнтами є цифровими.

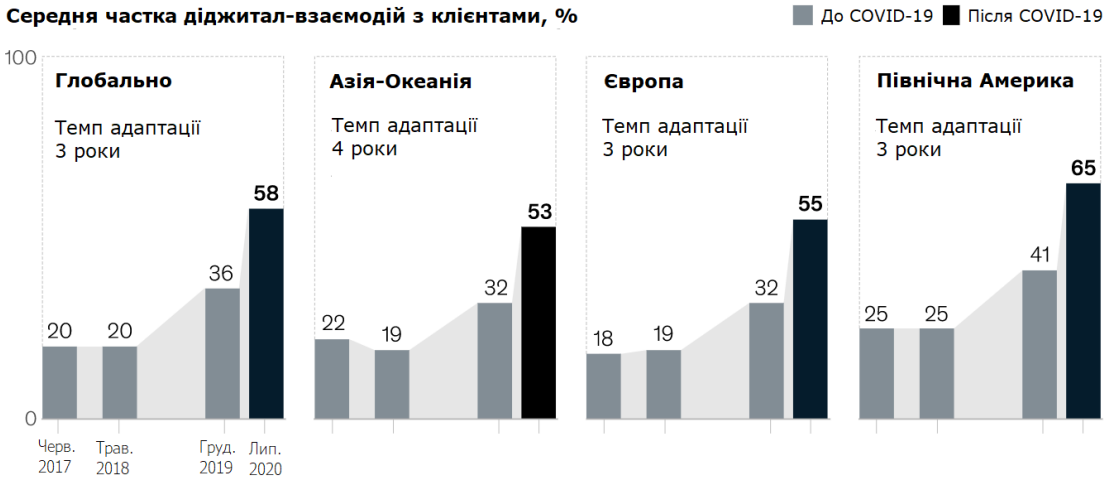


Рисунок 1 – Порівняння середньої частки цифрових взаємодій з клієнтами в умовах пандемії

Безумовно, за великої кількості пропозицій на ринку, а в даному випадку доступних для споживачів веб-сторінок, з'являється та зростає конкуренція на стороні бізнесу. Постає питання важливості певних аспектів веб-сайту та різних сторін взаємодії з ним користувачів. Задля заохочування нових клієнтів бізнес намагається отримати високі рейтинги в пошукових двигунах таких як Google або Bing, а для утримання споживачів та збільшення прибутку може спробувати поліпшити якість контенту сторінок, полегшити способи споживання інформації та оптимізувати свій сайт. До того ж, з часом користування веб-сайтами користувачі стають більш вибагливими до швидкості, рівня оптимізації та функціоналу.

Метою дослідження є аналіз важливості певних аспектів веб-додатків та техніки покращення свого продукту для успіху бізнесу.

1 АНАЛІЗ ІНСТРУМЕНТІВ FRONTEND-РОЗРОБКИ ТА МЕТОДІВ ОПТИМІЗАЦІЇ ВЕБ-САЙТУ

1.1 Аналіз інструментів розробки frontend-застосунків

Розробка фронтенду полягає у розробці візуальних та інтерактивних елементів веб-сайту, з якими користувачі взаємодіють безпосередньо. Це поєднання HTML, CSS і JavaScript, де HTML забезпечує структуру сторінки, CSS – стилі і макет, а JavaScript – інтерактивність і динамічну поведінку елементів сторінки. За час існування фронтенду було створено безліч інструментів направлених на покращення якості додатків, можливість надавати кращий та більш конкурентний функціонал користувачу, а також полегшення та пришвидшення роботи розробників. На сьогодні усі сучасні застосунки залучають певний набір технологій та інструментів, який доповнює та розвиває базові HTML, CSS та JavaScript.

Усі інструментальні засоби зазвичай обираються досвідченими спеціалістами в ролі архітектора або лідера команди на етапі планування IT-проєкту. Цей вибір зазвичай базується на вартості, строках, початкових вимогах та цілях проєкту. Він також визначає ключові направлення, якими можуть бути швидке завантаження та висока інтерактивність для залучення клієнтів бізнесу або значна мультифункціональність додатка, яка не потребує високої швидкості, натомість є корисним інструментом для внутрішніх членів компанії. Сучасні інструменти розробки frontend-додатків класифікуються наступним чином:

- фреймворк;
- стратегія рендерингу;
- CSS-препроцесор;
- збірник модулів.

1.1.1 Опис призначення frontend-фреймворків

Frontend-фреймворки – це інструменти розробки програмного забезпечення, які використовуються командами розробників для створення інтерфейсів користувача. Вони надають набір заздалегідь написаного коду, який можна повторно використовувати, дотримуючись певних стандартів, які зазвичай включають конвенції для організації різних частин коду. Дотримуючись цих стандартів, зовнішні фреймворки допомагають розробникам легше розуміти та працювати з кодом, написаним іншими, що робить їх особливо корисними, коли різні розробники працюють над тією самою кодовою базою. Це гарантує, що ваші зовнішні розробники застосовують єдиний набір правил, документацію та передові практики для написання коду. Якщо IT-проєкт втрачає одного з розробників, простіше знайти заміну без зайвого часу на підготовку, оскільки кожен розробник відразу розуміє, як виконувати ту ж роботу однаково.

Використовуючи чистий JavaScript замість зовнішнього фреймворку, розробники повинні були б створювати власну комплексну структуру для організації та реалізації функцій у навіть невеликий веб-додатках. Такий підхід займає велику кількість часу та дуже обмежує команду у функціоналі, який можна реалізувати для бізнесу.

Вибір фреймворку заздалегідь часто обумовлений цілями проєкту, вартістю, строком, а також можливістю формування команди з розробників з певними навичками. Інколи компанії вимушені підлаштовуватися під технологічний стек, яким володіє наявна команда.

1.1.2 Опис ролі стратегії рендерингу

Одним із основоположних рішень, які приймаються під час планування IT-проєкту з розробкою веб-додатка, є вибір стратегії рендерингу. Веб-додаток може будуватися на стороні сервера, клієнта, частково на обох сторонах, або ж статично зберігати на сервері. Сторінки веб-сайту можуть оновлюватися за кожним запитом, зберігатися в кеш-пам’яті протягом певного періоду часу або залишатися статичними постійно.

Вибір стратегії базується на бізнес-вимогах, ролі продуктивності веб-сайту, важливості SEO та не повинен залежати від фреймворку, який використовується для розробки. Вибір найбільш підходящої стратегії рендерингу може призвести до швидшого відтворення веб-сайту та чудової продуктивності завантаження за низьких витрат на обробку. З іншого боку, невдалий вибір стратегії може зробити додаток неефективним та ресурсно-затратним для бізнесу.

Коректно обрана стратегія може значно змінити досвід розробника (DX), коли додаток створюється для команди інженерів або внутрішніх членів компанії, де швидке завантаження додатка або SEO не є ключовими вимогами, і досвід користувача (UX), якщо він націлений на кінцевих користувачів.

1.1.3 Призначення CSS-процесору

Препроцесори CSS компілюють код, який написаний CSS-подібною мовою за допомогою спеціального компілятора. Препроцесори CSS дедалі більше стають основою робочого процесу веб-розробників інтерфейсу. Зазвичай реалізація комплексних ідей пов’язаних із стилями використовуючи «чистий» CSS є об’ємною та складною задачею. Намагаючись полегшити її

використання, розробники часто звертаються до використання препроцесорів, таких як SCSS, SASS або LESS.

Препроцесори додають більш логічний синтаксис і інструменти, такі як змінні, оператори if/else і цикли. Це робить CSS більш ефективним і лаконічним, потужним і динамічним. Використовуючи препроцесор CSS, розробник може написати складніший варіант стилів, а вихідний код може бути коротшим і легшим для читання. Більшість сучасних проєктів з frontend-розробкою запускаються з використанням препроцесора та включають в себе вибір даного інструменту заздалегідь, наприклад на етапі планування.

1.1.4 Роль збірників модулів у frontend-проєктах

Збірник модулів є інструментом розробки, який об’єднує кілька JavaScript-файлів в один, який завантажується браузером з серверу в додатках production-версії. Головні особливості сбірника полягають у поділі коду (code-splitting), що дає змогу ділити код програми на керовані блоки, які можна завантажувати за запитом, тобто JavaScript-код оброблюється браузером швидше, а користувачі мають змогу взаємодіяти зі сторінкою набагато раніше, ніж якби їм доводилося чекати, поки завантажиться та активується увесь код додатку.

Також однією з ключових функцією є генерування графіка залежностей, коли збірник аналізує, ділить та поміщає код у певну кількість модулів. Це означає, що збирач модулів відстежує залежності файлів проєкту і бібліотек, які залучаються, починаючи з наданої точки входу. Цей графік залежностей гарантує, що весь вихідний код і пов’язані файли без помилок і постійно оновлюються. Найбільш вживаними збірниками на даний момент є Webpack та Rollup.

Такі збірники модулів автоматично видаляють код, який не використовується, за допомогою техніки tree-shaking, мінімізують код та стискають файли для підвищення швидкості обробки коду браузером, а також ефективно розміщують та завантажують статичні медіа-файли (зображення, відео, тощо).

1.2 Аналіз важливості швидкості та продуктивності веб-сайту

Велика конкуренція на цифровому ринку та вибагливість користувачів випливають у наявні високі вимоги до швидкості веб-сторінок. Низька швидкість сайту може призвести до поганого рейтингу в пошуковій системі, зниження загального трафіку сайту та негативного досвіду користувачів. Ці чинники на сьогодні вважаються критичними та їх нерозвиненість призводить до втрати потенційних або ж наявних користувачів.

Дослідження показують, що швидкість сайту напряму впливає на коефіцієнт конверсії – швидкість, з якою користувачі виконують бажану дію. Як результат, не лише більше користувачів залишаються на сайтах, які швидко завантажуються, але й швидкість конверсії більша, ніж на повільніших сайтах.

Дослідження також виявили негативний вплив поганої продуктивності на бізнес-цілі. Наприклад, одна з найбільших телевізійних мереж світу BBC з’ясувала, що вони втрачали додаткових 10% користувачів за кожну додаткову секунду завантаження сайту [2].

Ще одним важливим пунктом є те, що велика частка користувачів продовжує відвідувати сторінки зі старих пристроїв або/та мають слабке інтернет-з'єднання. За даними досліджень понад 60% інтернет-трафіку на березень 2023 проходить через мобільні пристрої, при цьому більш ніж 92% користувачів отримують доступ до інтернету з мобільного телефону [3]. В таких умовах, навіть веб-сайт з середніми показниками швидкості на десктоп-

пристроях та при швидкому інтернеті на мобільних пристроях значно втрачатиме свою ефективність та мету.

1.3 Аналіз проблем та ключових факторів продуктивності сучасних веб-сайтів

1.3.1 SEO

Розглянемо поширені проблеми та недоліки веб-сайтів, що стосуються продуктивності сторінок та ефективності при взаємодії з користувачами.

Прагнучи залучити якомога більше потенційних клієнтів на свій сайт, бізнес розглядатиме необхідність високих рейтингів в пошукових системах. Незважаючи на усі позитивні сторони сторінки, від оптимізованості до якості наповнення – усе це є марним якщо споживач не відвідав веб-сайт.

Пошукова оптимізація SEO є одним з ключових чинників успішності та ефективності веб-сайту. Вона впливає на:

– видимість сайту через пошук;

– залучення більшої кількості трафіку;

– укріплення бренду та підвищення пізнаванності компанії чи організації;

– досягнення бізнес-цілей та успіху в конкурентній боротьбі.

За даними Google на показники впливають наступні чинники:

– якість наповнення (контенту) – наскільки добре інформація на сторінці відповідає на пошуковий запит користувача;

– структура сторінки – чіткі та зрозумілі заголовки, корисні допоміжні заголовки розділів, альтернативні (alt) теги для зображень, зрозуміла ієрархія сторінки, тощо;

– користувальницький досвід (UX) – враження користувача від взаємодії з веб-сайтом, включаючи швидкість завантаження, відгуки, функціональність.

Останній пункт демонструє, що рейтинги SEO залежать і від технічних показників сторінки. У 2010 році Google оголосив, що швидкість сайту буде враховуватися в рейтингу пошуку [4]. З того часу продуктивність сайту стала більш впливовою, і оптимізація продуктивності є значною частиною оптимізації для SEO. У своєму початковому оголошенні швидкості сайту як фактора рейтингу Google наголошував на перспективі користувача та на тому, як повільний веб-сайт негативно впливає на досвід користувача. Google також зазначив, що користувачі рідше залишаються на веб-сайтах, які завантажуються повільно.

Зручність використання веб-сайту та загальна взаємодія з користувачем, є вирішальними, особливо для мобільних пристроїв, і власні дослідження Google показали, що 53% мобільних сайтів взагалі закриваються, якщо вони завантажуються довше 3 секунд [4]. Саме тому, оптимізованість сторінок для мобільних пристроїв є особливо важливою для SEO, а загалом вона тісно пов’язана та впливає на показники SEO.

Як приклад, проведена оптимізація та перебудова сторінок соціальної мережі Pinterest у 2020 році призвела до скорочення часу очікування на 40%, збільшення SEO-трафіку на 15% і збільшення коефіцієнта конверсії для реєстрації на 15% [5].

1.3.2 Швидкість завантаження сторінки

Повільне завантаження сторінки може змусити відвідувача покинути ваш веб-сайт, можливо, ще до того, як вміст завантажиться, щоб шукати інформацію в іншому місці. У 2017 році Google та SOASTA провели спільне дослідження стану ефективність роздрібної торгівлі в Інтернеті [6]. Проаналізувавши його дані можна побудувати та дослідити залежність вірогідності втрати користувача від часу завантаження сторінки (таблиця 1.1).

Таблиця 1.1 – Залежність вірогідності втрати користувача від часу завантаження сторінки

Збільшення часу завантаження сторінки	Ймовірність втрати користувача
З 1 секунди до 3 секунд	32%
З 1 секунди до 5 секунд	90%
З 1 секунди до 6 секунд	106%
З 1 секунди до 10 секунд	123%

Інша частина дослідження вказує на те, що швидші сторінки змушують покупців проводити більше часу на сайтах роздрібної торгівлі, відвідувати більше сторінок і додавати більше товарів у кошики. Тривалість сеансу розглядається разом із такими показниками, як показник покидання сайту і конверсії (в дослідженні визначалися як кількість сторінок, відвіданих за одне відвідування) є сильним показником залученості та задоволеності користувачів.

Подібно до впливу уповільнення на показник покидання сайту, приріст у 100 мілісекунд практично не вплинув на тривалість сеансу. Однак на одній і двох секундах ефект був значним. Сеанси із середньою тривалістю завантаження сторінки, яка була на секунду нижчою за оптимальну швидкість, також були на 25% коротшими. Затримка на 2 секунди корелювала зі зменшенням тривалості сеансу на 51% для користувачів мобільних пристроїв, зменшенням на 47% для користувачів комп’ютерів і майже на 38% для користувачів планшетів (рисунок 1.1).

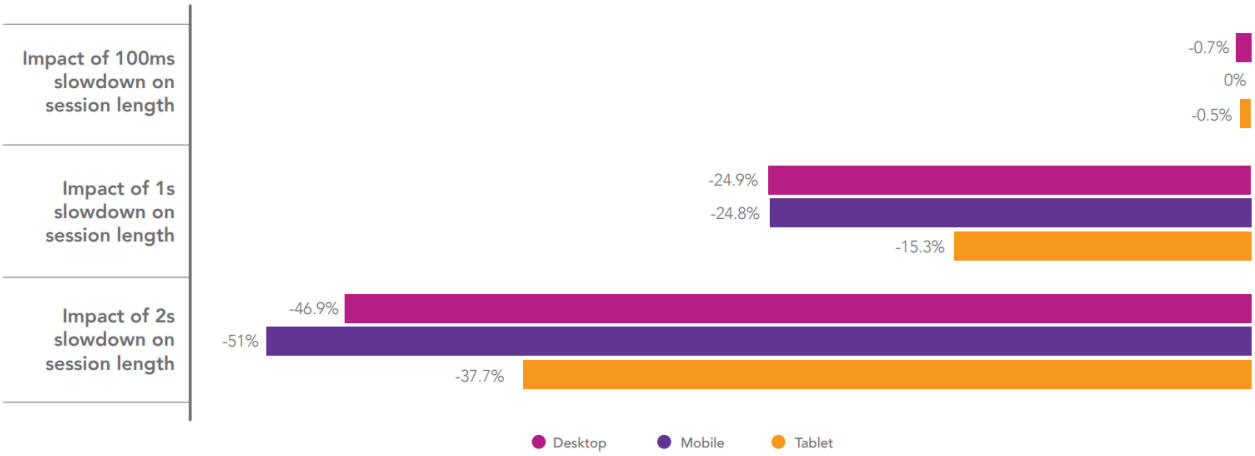


Рисунок 1.1 – Вплив сповільнення сторінки на тривалість сеансу за типом пристрою

Враховуючи наведені дані та вплив швидкості на рейтинги SEO, не залишається сумнівів щодо критичності цього фактору на сучасний бізнес.

1.3.3 Швидкість завантаження та візуалізації контенту й медіа сторінки

Великий вплив на рівень конверсії та користувальницький досвід також має швидкість відображення цікавого для споживача та важливого для бізнесу контенту: великі зображення в заголовках, рекламні банери, відео з автоматичним відтворюванням. Крім того, веб-сторінки з великим розміром файлів завантажуються довше, що має прямий вплив на вже розглянуті чинники швидкості рендерингу сторінки, показники SEO.

Швидка візуалізація корисної інформації, медіа, а також елементів взаємодії було запроваджено великими компаніями, внаслідок чого дослідження Google 2021 р. показують покращення показників конверсії, а також тривалості сесії користувача [7].

Так, наприклад, за даними дослідження:

– найбільша комерційна платформа Індонезії Tokopedia досягла на 23% кращу середню тривалість сеансу завдяки покращенню часу візуалізації найбільших файлів 6-ти основних сторінок веб-сайту на 55%;

– індійська компанія Nykaa виявила, що покращення швидкості візуалізації зображень на 40% призвело до збільшення органічного трафіку з міст T2/T3 на 28%;

– платформа онлайн-продажу Cdiscount покращила показники швидкості доступності сторінки для взаємодії та візуалізації контенту, що сприяло збільшенню доходу на 6% від дні розпродажу;

– Vodafone Italy покращила показник візуалізації найбільшого медіа-контенту на 31% та збільшила продажі на 8%, спостерігала перехід до кошика частіший на 11%, та кількість загальних відвідувань на 15% (рисунок 1.2).

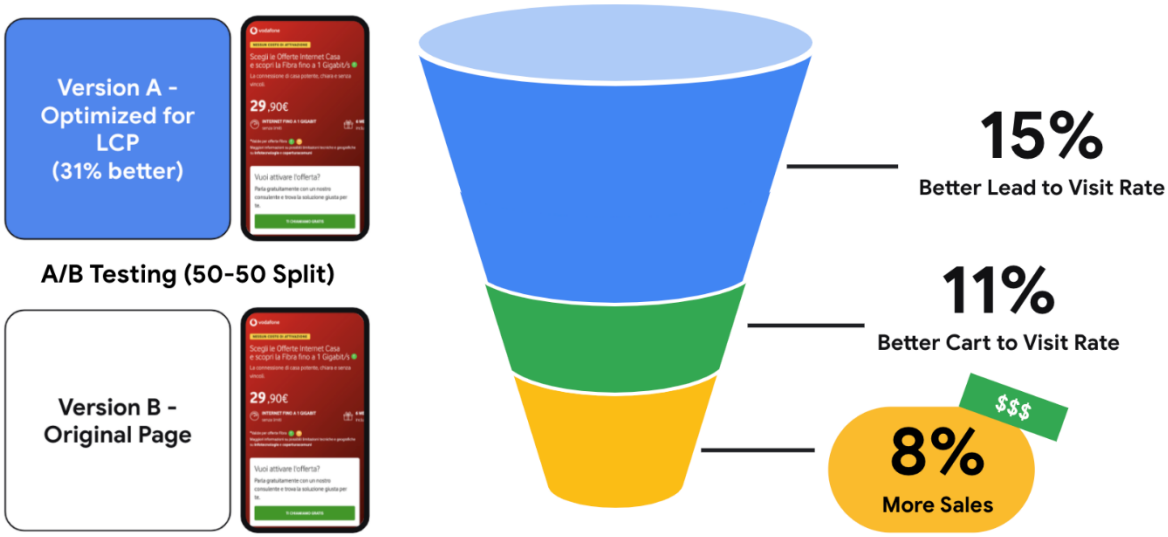


Рисунок 1.2 – Результати оптимізації веб-додатку Vodafone Italy

1.4 Огляд методів оптимізації, показників продуктивності, видів рендерингу веб-сторінок

1.4.1 Типи рендерингу (будування) сторінок

На сьогодні для розробки веб-додатків використовується декілька базових фреймворків без вагомої переваги у одного з них. Зазвичай розробники акцентують увагу на одному з фреймворків та покращують свої знання і навички не витрачаючи сили на одразу декілька. Кожен з фреймворків має інколи вкрай різну архітектуру та техніки оптимізації, що робить задачу загального методу оптимізації для більшості сайтів неможливою. Однак, кожен веб-сайт може реалізовувати різні техніки рендерингу своїх сторінок. Існує декілька поширених типів рендерингу зі своїми переваги та недоліками, які можна використовувати на свою користь залежно від вимог та цілей. Одними з основних типів будування сторінки:

− Server-Side Rendering – відтворення клієнтської або універсальної програми в HTML на сервері;

− Client-Side Rendering – рендеринг програми в браузері, як правило, за допомогою DOM;

− Rehydration – «завантаження» коду JavaScript на стороні клієнта (зазвичай браузеру) таким чином, щоб він повторно використовував побудоване на сервері дерево HTML DOM і дані;

− Prerendering – запуск додатку на стороні клієнта під час створення, щоб зафіксувати її початковий стан як статичний HTML.

1.4.2 Показники продуктивності та інструменти їх аналізу

Для вимірювання показників SEO, для дослідження продуктивності веб-сайту, а також під час запису та аналізу показників завантаження та UX характеристик сторінки – Google використовує Web Vitals (веб-показники), щоб визначити, наскільки швидко та плавно завантажуються елементи на веб-сторінці та, у свою чергу, як користувачі взаємодіють із сайтом [8]. Web Vitals – ініціатива Google, спрямована на надання уніфікованих інструкцій щодо сигналів якості, необхідних для забезпечення чудової взаємодії з користувачем в Інтернеті. Вони є кількісними показниками, які роблять аналізування продуктивності сторінок більш чітким, зрозумілим та легким в порівняннях.

Також, в 2020 Google виділила три основні веб-показники та назвала їх Core Web Vitals. Кожен з основних веб-показників представляє окремий аспект взаємодії з користувачем, що відображає реальний досвід критично-орієнтованого на користувача результату. Вибір саме цих показників показує зосередженість на трьох аспектах взаємодії з користувачем – завантаженні, інтерактивності та візуальній стабільності. Для зручного аналізу сторінок Google надає доступні інструменти:

− Chrome User Experience Report (звіт Chrome про взаємодію з користувачем) – збирає анонімні дані реальних користувачів для кожного Core Web Vital;

− PageSpeed Insights – звітує про взаємодію користувача зі сторінкою як на мобільних, так і на настільних пристроях, а також надає пропозиції щодо того, як цю сторінку можна покращити;

− Lighthouse – автоматизований інструмент, який перевіряє продуктивність, доступність сторінок, прогресивні веб-додатки (PWA), SEO тощо.

Розглянуті метрики та інструменти аналізу допоможуть зрозуміти слабкі сторони вже готової сторінки та обрати шляхи оптимізації веб-сайту. При вірному запровадженню, вони покращать показники швидкості, користувальницький досвід, SEO, рівень конверсії сайту.

Ще одним інструментом вирішення задачі покращення продуктивності сторінок може бути performance-методологія RAIL. Response, Animation, Idle, Load – модель, орієнтована на користувача, яка забезпечує структуру для сприйняття продуктивності. Вона розбиває досвід користувача на ключові дії (наприклад, натискання, прокручування, завантаження) і допомагає визначити цілі продуктивності для кожної з них.

1.5 Опис постановки задачі дослідження

Задача дослідження полягає в аналізі розглянутих методів для досягнення цілей та відповідальності вимогам сучасних веб-сайтів. Враховуючи наведені проблеми та наявні поширені слабкі сторони необхідно виявити найбільш дієві методології та техніки для будування або оптимізації сайтів за усіма розглянутими важливими критеріями, що врешті допоможе перемогти конкуренції та досягти поставлених цілей бізнесу.

2 ОПИС ПОКАЗНИКІВ ПРОДУКТИВНОСТІ І СТРАТЕГІЙ РЕНДЕРИНГУ З ВИЯВЛЕННЯМ ПЕРЕВАГ ТА НЕДОЛІКІВ

Протягом багатьох років Google надавав багато інструментів власникам та розробникам веб-сайтів для вимірювання ефективності. Web Vitals – показники якості для забезпечення кращого користувальницького досвіду в мережі – використовуються розробниками найбільше для аналізу та оптимізації. В 2020 році в компанії вирішили спростити інструменти та методи аналізу з метою полегшення вивчення та використання їх усіма зацікавленими особами, не тільки розробниками з безпосереднім досвідом. Таким чином, компанія запроваджує поняття Core Web Vitals та допомагає сайтам зосередитися на найважливіших показниках.

Кожен з основних веб-показників (CVW) представляє окремий аспект взаємодії з користувачем, його можна виміряти на реальному веб-сайті і він відображає реальний досвід критично орієнтованого на користувача результату. На даний момент набір ключових показників фокусується на трьох аспектах взаємодії з користувачем – завантаженні, інтерактивності та візуальній стабільності – і включає такі показники:

– Largest Contentful Paint (LCP): означає малювання найбільшого вмісту сторінки та вимірює ефективність завантаження. Щоб забезпечити хорошу взаємодію з користувачем, LCP має відбуватися протягом 2,5 секунд після початку завантаження сторінки;

– First Input Delay (FID): позначає час очікування користувача до першої взаємодії з контентом та вимірює інтерактивність. Щоб забезпечити хорошу взаємодію з користувачем, сторінки повинні мати FID 100 мілісекунд або менше;

– Cumulative Layout Shift (CLS): означає сукупний зсув макета та вимірює візуальну стабільність. Щоб забезпечити хорошу взаємодію з користувачем, сторінки повинні підтримувати CLS 0,1. або менш.

Для кожного з наведених вище показників, щоб переконатися, що рекомендована ціль досягається для більшості користувачів, хорошим порогом для вимірювання є 75-й процентиль завантажень сторінки, сегментований за мобільними та настільними пристроями.

Інструменти визначення показників веб-сторінки можна поділити на 2 типи: польові (аналіз реальних сторінок) та лабораторні (аналіз під час розробки). До польових інструментів можна віднести наступні: Chrome User Experience Report, PageSpeed Insights, Search Console (Core Web Vitals report). Усі вони мають підтримку трьох ключових показників тому добре пасуватимуть розробнику. Chrome User Experience Report збирає анонімні дані реальних користувачів для кожного Core Web Vital. Ці дані дають змогу власникам сайтів швидко оцінювати їх ефективність, не вимагаючи від них вручну інструментувати аналітику на своїх сторінках, а також забезпечують такі інструменти, як PageSpeed Insights і «Core Web Vitals report» від Search Console.

Дані, надані у звіті про взаємодію з користувачем браузеру Chrome, пропонують швидкий спосіб оцінити продуктивність сайтів. Таку перевагу компенсує відсутність детальної телеметрії за переглядом сторінки, яка часто необхідна для точної діагностики, моніторингу та швидкого реагування на регресії. Ефективним вирішенням такої задачі може стати налаштування власного моніторингу реальних користувачів розробниками сайтів, наприклад, використовуючи бібліотеку «web-vitals» для JavaScript.

Хоча всі основні веб-показники є, перш за все, польовими показниками, багато з них також можна виміряти в процесі розробки. Лабораторні вимірювання – це найкращий спосіб перевірити продуктивність нового функціоналу під час розробки до того, як вони будуть випущені для користувачів. Це також найкращий спосіб виявити регресію продуктивності до того, як вона станеться. Для вимірювання основних веб-показників у лабораторному середовищі можна використовувати інструменти Chrome Dev Tools та Lighthouse, які вбудовані в браузер Chrome за замовчуванням.

Проте такі інструменти завантажують сторінки в симульованому середовищі без користувача, тому не можуть виміряти FID (користувач не вводить дані). Однак загальний час блокування (Total Blocking Time – TBT) можна виміряти в лабораторії та є прийнятною альтернативою для FID. Оптимізація продуктивності, яка покращує TBT у процесі розробки, має покращити FID у польових умовах.

2.1 Largest Contentful Paint

Метрика Largest Contentful Paint (LCP) повідомляє проміжок часу великого зображення або текст блокування, що відображається в viewport, порівняно з тим, як сторінка перша завантажена. Щоб забезпечити хорошу взаємодію з користувачем, сайти повинні прагнути мати результат LCP в 2.5 секунди або менше. При показниках від 2.5 секунд до 4 секунд рекомендується провести оптимізацію сторінки. Результат в 4 секунд вважається критичним та має значний негативний вплив на користувальницький досвід.

На рисунку 2.1 зображено ефективність показників LCP.

Рисунок 2.1 – Ефективність показників LCP

Дослідивши документацію Largest Contentful Paint API, можна зробити висновок, що для вимірювання LCP розглядаються такі типи елементів:

– елементи <img>;

– елементи <image> всередині елемента <svg>;

– елементи <video> із зображенням постеру (poster image) (використовується час завантаження зображення постеру);

– елемент із фоновим зображенням, завантаженим через функцію «url()» (на відміну від градієнта CSS);

– елементи рівня блоку, що містять дочірні текстові елементи вбудованого рівня.

LCP можна виміряти під час розробки та на реальних веб-сайтах, тому усі згадані інструменти аналізу підходять для даної задачі. Оптимізувати показники LCP можна використовуючи техніки описані нижче.

Усунення затримки завантаження ресурсу (LCP) – необхідно переконатися, що ресурс LCP починає завантажуватися якомога раніше, і як найкраще, ресурс повинен почати завантажуватися одночасно з першим ресурсом, завантаженим цією сторінкою. Сканер попереднього завантаження браузера виявляє LCP у документі HTML, що сприяє швидшому завантаженню, тільки у випадках:

– LCP є елементом «img», і його атрибути «src» або «srcset» присутні у початковій розмітці HTML;

– для елемента LCP потрібне фонове зображення CSS, але це зображення попередньо завантажується через тег «link» з атрибутом «rel="preload"» у розмітці HTML;

– LCP – текстовий елемент, для візуалізації якого потрібен веб-шрифт, і шрифт завантажується через тег «link» з атрибутом «rel="preload"» у розмітці HTML;

– варто також зберігати ресурс на домені веб-сайту, бо в інакшому випадку браузеру необхідно буде під’єднатися до іншого домену перед тим, як зробити запит.

Усунення затримки візуалізації (рендерингу) елемента – важливо переконатися, що елемент LCP може бути побудований одразу після завершення завантаження його ресурсу, незалежно від того, коли це станеться.

Основна причина, через яку елемент LCP не зможе відобразити одразу після завершення завантаження ресурсу, полягає в тому, що відтворення заблоковано з іншої причини:

– відтворення всієї сторінки заблоковано через таблиці стилів або синхронні сценарії в тегу head, які все ще завантажуються;

– ресурс LCP завершив завантаження, але елемент LCP ще не додано до DOM (він очікує на завантаження коду JavaScript);

– елемент приховано якимось іншим кодом, наприклад бібліотекою тестування A/B, яка все ще визначає, у якому експерименті має брати участь користувач;

– основний потік заблоковано через довготривалі завдання, і робота візуалізації має зачекати, поки ці завдання завершаться.

Щоб усунути найпоширеніші причини затримки будування непотрібних елементів, можна:

– використовувати вбудовані в HTML стилі CSS (автоматизується за допомогою сучасних сбірників коду таких як Webpack або Rollup);

– зменшити розмір файлів CSS стилів, наприклад за допомогою видалення непотрібного CSS, відкладання завантаження некритичного CSS, розподілення стилів по файлам за розмірами пристроїв;

– відкладати завантаження некритичного JavaScript-коду або використовувати вбудований JavaScript із блокуванням рендерингу;

– використовувати рендеринг на стороні серверу.

Також можна скоротити час завантаження ресурсу – цього можна досягнути використанням одного з наступних методів, а краще поєднанням усіх:

– використання зображення оптимального розміру;

– використання сучасних форматів зображень (наприклад, webp);

– компресія зображень;

– зменшення розміру веб-шрифтів;

– зменшення розміру початкового HTML-файлу.

2.2 First Input Delay

Час очікування до першої взаємодії з контентом (FID) є важливою метрикою, орієнтованою на користувача, для вимірювання швидкості реагування на завантаження, оскільки вона кількісно визначає досвід, який відчувають користувачі, намагаючись взаємодіяти зі сторінками, що не відповідають. Низький FID допомагає переконатися, що сторінка придатна для використання.

Перше враження користувачів про швидкість завантаження вашого сайту можна виміряти за допомогою First Contentful Paint (FCP). Але наскільки швидко сайт може малювати пікселі на екрані, це лише частина ефективності. Не менш важливим є те, наскільки швидко реагує сайт, коли користувачі намагаються взаємодіяти з контентом сторінки.

FID вимірює час від моменту, коли користувач вперше взаємодіє зі сторінкою (тобто, коли він намагається перейти за посиланням, натискає кнопку або використовує спеціальний елемент керування на основі JavaScript) до моменту, коли браузер фактично може почати обробку обробники подій у відповідь на цю взаємодію.

Щоб забезпечити позитивну взаємодію з користувачем, сайти повинні прагнути мати затримку першого введення 100 мілісекунд або менше. Щоб переконатися, що такі показники досягаються для більшості користувачів сайту, гарним порогом для вимірювання є 75-й процентиль завантажень сторінок, сегментований за мобільними та настільними пристроями.

На рисунку 3.2 зображено те, як доцільно розглядати та оцінювати показники затримки взаємодії з контентом. При затримці в більше 100 мілісекунд рекомендовано провести оптимізацію певної частини роботи сайту, щоб посприяти покращенню показника. Затримка в більше ніж 300 мілісекунд вважається поганою та має великий негативний вплив на конверсію нових користувачів сайту.

Рисунок 2.2 – Ефективність показників FID

Загалом затримка інтерактивності виникає через те, що головний потік браузера зайнятий чимось іншим, тому він (ще) не може відповісти користувачеві. Однією з поширених причин цього може статися те, що браузер зайнятий аналізом і виконанням великого файлу JavaScript, завантаженого вашою програмою. Поки він це робить, він не може запустити жодних прослуховувачів подій, оскільки JavaScript, який він завантажує, може наказати йому зробити щось інше.

FID вимірює лише "затримку" в обробці події. Він не вимірює ні час обробки події, ні час, потрібний браузеру для оновлення інтерфейсу користувача після запуску обробників подій. Хоча цей час справді впливає на взаємодію з користувачем, включення його як частини FID стимулюватиме розробників реагувати на події асинхронно, що покращить показники, але, ймовірно, погіршить роботу.

FID вимірює дельту між моментом отримання вхідної події та моментом наступного простою основного потоку. Це означає, що FID вимірюється навіть у випадках, коли слухач подій не зареєстрований. Причина полягає в тому, що багато взаємодій користувача не вимагають прослуховування подій, але вимагають, щоб основний потік був неактивним для запуску.

Наприклад, усі наведені нижче елементи HTML мають дочекатися завершення поточних завдань у головному потоці, перш ніж відповідати на дії користувача: input, textarea, checkbox, radio button, select, link (anchor).

FID – це показник, який можна виміряти лише в польових умовах, оскільки для взаємодії зі сторінкою потрібен реальний користувач. Для вимірювання можна використовувати інструменти:

– Chrome User Experience Report;

– PageSpeed Insights;

– Search Console (Core Web Vitals report);

– «web-vitals» JavaScript-бібліотека.

Враховуючи те, що FID не може бути перевірений на етапі розробки, можна розглянути показник Total Blocking Time (TBT). Ця метрика піддається лабораторному вимірюванню та, за результатами досліджень, добре корелює з FID у польових умовах, а також фіксує проблеми, які впливають на інтерактивність. Оптимізація, яка покращує TBT у лабораторії, також матиме вплив на FID для користувачів сайту.

Розглянувши принципи та залежності показника First Input Delay, можна вивести ряд технік щодо покращення даного показника:

– поділ довготривалих завдань головного потоку на менш тривалі та ресурсо-затратні;

– оптимізація сторінки для готовності до взаємодії;

– використання web worker;

– скорочення часу виконання JavaScript.

Довготривалі завдання – це періоди виконання JavaScript, коли користувачі можуть виявити, що інтерфейс користувача не реагує. Будь-який фрагмент коду, який блокує основний потік на 50 мілісекунд або більше, можна охарактеризувати як довготривале завдання. Такі завдання є ознакою потенційного роздування JavaScript (завантаження та виконання більше, ніж може бути потрібно користувачеві прямо зараз). Розподіл довгих завдань може зменшити затримку введення на сайті. FID помітно покращується при використанні технік code-splitting (поділі коду) та поділу довготривалих завдань.

Web workers дозволяють запускати JavaScript у фоновому потоці. Перенесення операцій, не пов’язаних з інтерфейсом користувача, до окремого робочого потоку може скоротити час блокування основного потоку та, як наслідок, покращити FID.

Обмеження кількості JavaScript на сторінці сайту зменшує кількість часу, який браузеру потрібно витрачати на виконання коду JavaScript. Це пришвидшує швидкість реакції браузера на будь-які дії користувача. Щоб зменшити кількість JavaScript, що виконується на сторінці можно вдатися до: відкладання невикористаного JavaScript, мінімізації невикористаних поліфілів (polyfills).

2.3 Cumulative Layout Shift

Cumulative Layout Shift (CLS) – вимірювання результатів найбільшого прояву зсуву (зміни, переміщення) макета сторінки для кожного несподіваного зміщення макета, яке відбувається протягом усього часу сесії користувача на сторінці.

Зсув макета відбувається кожного разу, коли видимий елемент змінює своє положення від одного візуалізованого кадру до іншого.

Серія змін макета, відома як вікно сеансу, – це коли одне або кілька окремих змін макета відбуваються у швидкій послідовності з менш ніж 1 секундою між кожною зміною та максимум 5 секундами для загальної тривалості вікна. Найбільша серія– це вікно сеансу з максимальним кумулятивним балом усіх змін макета в цьому вікні.

Частка впливу вимірює, як нестабільні елементи впливають на область перегляду між двома кадрами. Об’єднання видимих областей усіх нестабільних елементів для попереднього кадру та поточного кадру – як частка від загальної площі вікна перегляду – є часткою впливу для поточного кадру.

На рисунку 2.3 зображено елемент, який займає половину вікна перегляду в одному кадрі. Потім у наступному кадрі елемент зсувається вниз на 25% висоти вікна перегляду. Червоний пунктирний прямокутник вказує на об’єднання видимої області елемента в обох кадрах, яка в цьому випадку становить 75% від загального вікна перегляду, тому його частка впливу становить 0.75.

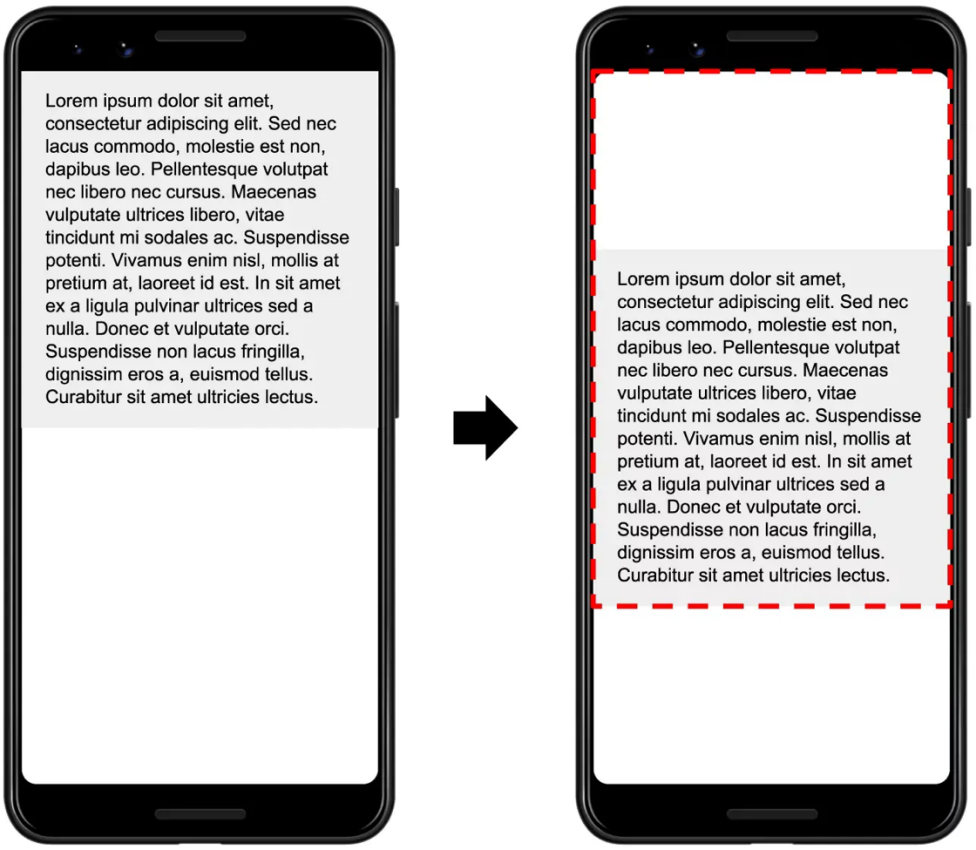


Рисунок 2.3 – Приклад зсуву макета

Інша частина рівняння оцінки зсуву макета вимірює відстань, на яку перемістилися нестабільні елементи відносно вікна перегляду. Частка відстані – це найбільша відстань, яку будь-який нестабільний елемент перемістив у кадрі (горизонтально чи вертикально), поділена на найбільший розмір вікна перегляду (ширину чи висоту, залежно від того, що більше).

У наведеному вище прикладі найбільшим розміром вікна перегляду є висота, а нестабільний елемент перемістився на 25% від висоти вікна перегляду, що робить частку відстані 0.25.

Отже, у цьому прикладі частка впливу становить 0.75, а частка відстані – 0.25, тому оцінка зсуву макета становить 0.75 * 0.25 = 0.1875.

CLS можна виміряти в тестових або в польових умовах, і це доступно в усіх згаданих вище інструментах.

Найпоширенішими причинами поганого CLS є:

– зображення (теги image) без розмірів;

– оголошення, вставки та iframe без розмірів;

– динамічно-доданий на сторінку контент;

– дії, які очікують відповіді мережі перед оновленням DOM.

Детально розглянувши та проаналізувавши метрики продуктивності веб-сайтів, доречно перейти до дослідження методів та методологій за допомогою яких сайти та їх сторінки будуються. Існує декілька різних способів будування і те, як реалізується логіка та рендеринг у веб-додатку є одним з ключових рішень в розробці, яке впливає на подальший успіх. Для кращого розуміння архітектури, яка обирається для розробки, детально розглядається кожен підхід, його особливості та відмінності від інших.

2.4 Server-Side Rendering

Візуалізація на стороні сервера – або ж Server-Side Rendering – генерує повний HTML для сторінки на сервері по запиту браузера. Це дозволяє уникнути додаткових дій для отримання даних і елементів розмітки на стороні клієнта, оскільки вони обробляються до того, як браузер отримає відповідь.

Server-Side Rendering зазвичай має швидкий First Contentful Paint. Також, запуск логіки сторінки та візуалізація на сервері дає змогу уникнути надсилання клієнту JavaScript-коду великого об’єму, що допомагає досягти швидкого Time To Interact. Це означає значну перевагу, оскільки за допомогою

серверного рендерингу насправді просто надсилається текст й посилання в браузер користувача.

На рисунку 3.1 графічно зображено приклад запиту сторінки – GET-запит до адреси «/top/1» із генерацією частини всього JS-коду на сервері, що призводить до швидкого FCP, та подальша обробка JS-коду на стороні клієнта, по завершенню якої, маємо задовільні результати з TTI.

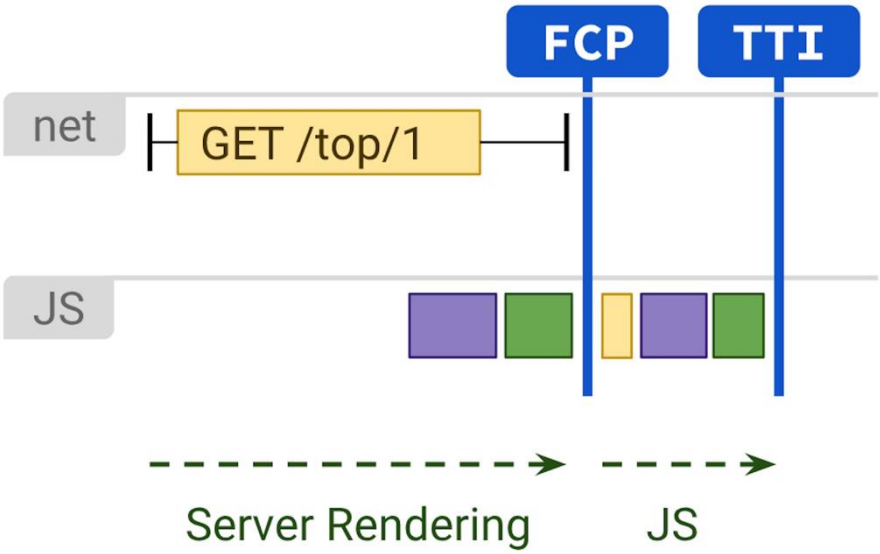


Рисунок 2.4 – Візуалізація підходу Server-Side Rendering

Цей підхід може добре працювати для широкого спектру пристроїв і умов мережі, а також дає можливість важливим методам оптимізації, наприклад для SEO. Оскільки під час запиту сторінки всі метадані вже було відображено в HTML, веб-сканери бачитимуть усі дані з програми, що ідеально підходить для видимості SEO.

Завдяки рендерингу на стороні сервера користувачі не чекатимуть обробки JavaScript, пов’язаного з процесором, перш ніж вони зможуть використовувати сайт. Навіть якщо стороннього JS неможливо уникнути, використання серверного рендерингу для зменшення власних витрат на JS може збільшити кількість ресурсів для решти задач. Однак у цього підходу є один основний недолік: генерація сторінок на сервері вимагає часу, що часто може призвести до поганих результатів метрики Time to First Byte.

Деякі сайти успішно застосували гібридні методи візуалізації. Сервер Netflix візуалізує свої відносно статичні головні сторінки, попередньо завантажуючи JS для сторінок із інтенсивною взаємодією, що дає цим важчим сторінкам, які будуються клієнтом, кращі шанси на швидке завантаження.

Багато сучасних фреймворків, бібліотек і архітектур дозволяють відтворювати ту саму програму як на клієнті, так і на сервері. Ці методи можна використовувати для серверного рендерингу, однак важливо зазначити, що архітектури, де рендеринг відбувається як на сервері, так і на клієнті, є окремим класом рішень із дуже різними характеристиками продуктивності та компромісами. Користувачі React можуть використовувати «renderToString()» або рішення, побудовані на його основі, наприклад Next.js, для серверного рендерингу. Користувачі Vue можуть переглянути посібник із рендерингу сервера Vue або Nuxt, у Angular є Universal. Однак у більшості популярних розчинів використовується певна форма rehydration (завантаження коду JavaScript на стороні клієнта таким чином, щоб він повторно використовував побудоване на сервері дерево HTML DOM і дані).

2.5 Static Rendering – Static Site Generation (SSG)

Статичний рендеринг відбувається під час збірки що призводить до швидкої візуаліції перших пікселів (First Paint), гарних результатів FCP та Time To Interactive – припускаючи, що кількість JS на стороні клієнта обмежена. На відміну від Server-Side Rendering, при SSG вдається досягти постійно швидкого Time to First Byte, оскільки HTML для сторінки не потрібно генерувати на при кожному запиті клієнта. Як правило, статичний рендеринг означає завчасне створення окремого файлу HTML для кожної URL-адреси. Завдяки HTML-відповідям, які генеруються заздалегідь, статичні візуалізації можна розгортати в кількох CDN, щоб скористатися перевагами техніки

кешування даних «edge-caching» – використання серверів кешування для зберігання вмісту доступнішому для користувачів.

На рисунку 3.2 графічно зображено приклад запиту вже сгенерованої сторінки на сервері, яка прийде на клієнт у вигляді статичного HTML.

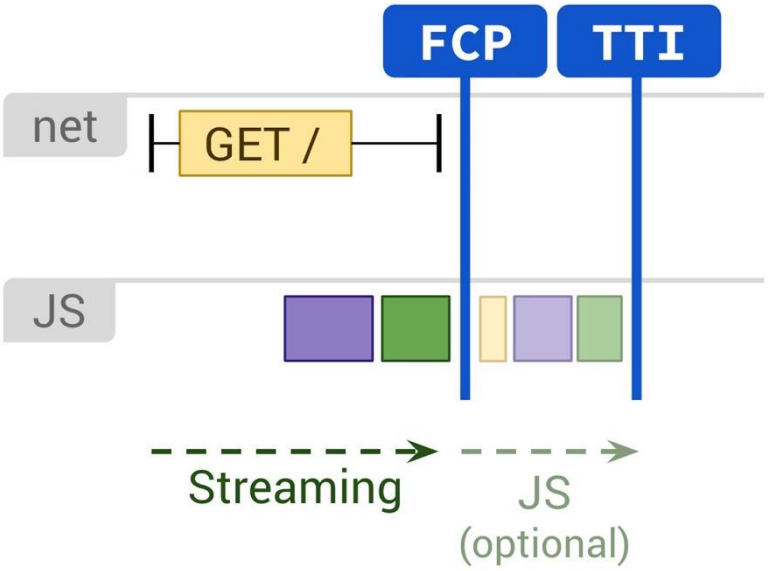


Рисунок 2.5 – Візуалізація підходу Static Rendering

Одним із недоліків статичного рендерингу є те, що для кожної можливої URL-адреси необхідно створювати окремі файли HTML. Це може бути складним або навіть нездійсненним, якщо ви не можете заздалегідь передбачити, якими будуть ці URL-адреси, або для сайтів із великою кількістю унікальних сторінок.

Для користувачів React доступні технології Gatsby, статичний експорт Next.js або Navi – усе це робить зручним створення веб-компонентів. Однак важливо розуміти різницю між статичним рендерингом і попереднім рендерингом (prerendering): статично-збудовані сторінки є інтерактивними і не потребують виконання великої кількості JS-коду на стороні клієнта, тоді як попередній рендеринг покращує First Paint або FCP одно-сторінкової програми, яку потрібно завантажувати на клієнті, щоб сторінки були справді інтерактивними.

Для перевірки того, чи даний веб-додаток є статичним чи збудований з prerendering, можна вимкнути JavaScript і завантажити створені веб-сторінки. Для статично відтворених сторінок більшість функціональних можливостей існуватимуть без увімкнення JavaScript. Для попередньо відрендерених сторінок все ще можуть працювати деякі базові функції, як наприклад посилання, але більша частина сторінки буде недоступна для взаємодії.

Іншим корисним тестом є уповільнення мережі за допомогою Chrome DevTools і спостереження за тим, скільки JavaScript було завантажено, перш ніж сторінка стане інтерактивною. Попереднє відтворення зазвичай потребує більше JavaScript-коду, щоб стати інтерактивним, і цей код має тенденцію бути більш складним, ніж підхід прогресивного вдосконалення, який використовується під час статичного відтворення.

До переваг SSG можна віднести те, що:

– так само, як рендеринг на стороні сервера, сторінки сгенеровані методом SSG доступні одразу й не потребують додаткових даних з серверу;

– SSG також чудово підходить для SEO, оскільки HTML створюється перед надсиланням клієнту, видимість для роботів SEO є ідеальною за допомогою SSG;

– час обслуговування статичних сторінок неймовірно швидкий, порівняно із часом надсилання порожньої сторінки, яку ми надсилаємо під час візуалізації на стороні клієнта (CSR).

Певні архітектурні особливості методу SSG попри значні переваги також випливають у наступні недоліки:

– час будування статичних сторінок може бути досить тривалим, на що впливає кількість сторінок веб-сайту та їх об’єм;

– дані завантажуються лише один раз під час створення, тому немає оновлення даних при повторних запитах на сторінку;

– недоступність глобального об’єкту window на стороні сервера, який може бути критично потрібним для певних задач.

2.6 Client-Side Rendering

Будування сторінок на стороні клієнта (CSR) означає їх рендеринг безпосередньо в браузері використовуючи JavaScript-код. Уся логіка, вибірка даних, створення шаблонів і маршрутизація обробляються на клієнті, а не на сервері.

Рендеринг на стороні клієнта може бути важким за ресурсами та повільним для мобільних пристроїв. Також, він може наблизитися до продуктивності чистого серверного рендерингу, якщо виконуватиме мінімальну роботу, зберігаючи жорсткий бюджет JavaScript і доставляючи ресурси за якомога менше затримок при запитах. Критичні сценарії та дані можуть бути доставлені з серверу швидше за допомогою <link rel=preload>, завдяки чому дані та медіа швидше обробляться. На рисунку 3.3 графічно зображено приклад реалізації методу CSR.

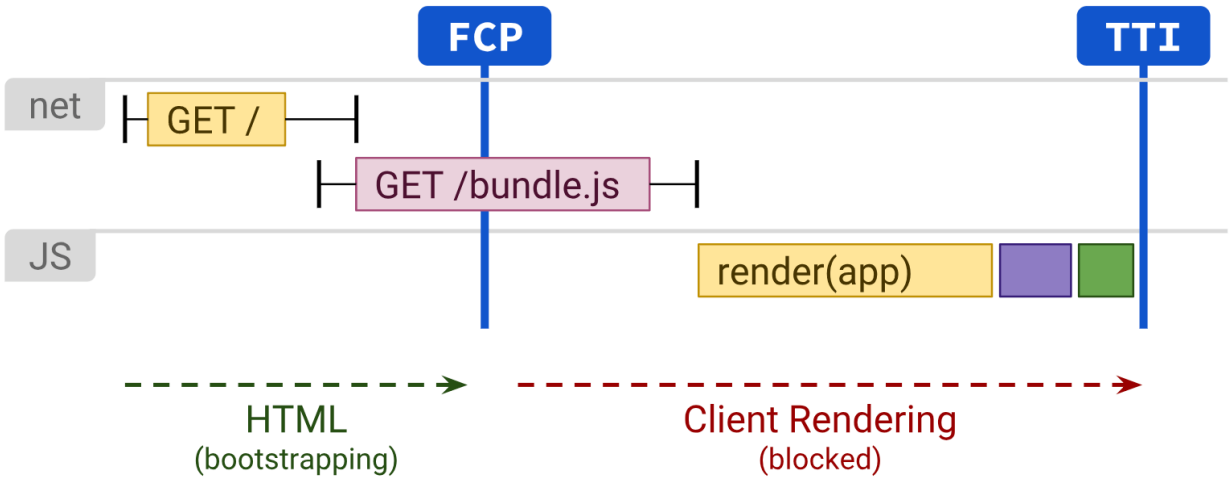


Рисунок 2.6 – Візуалізація підходу Client-Side Rendering

Основним недоліком рендеринга на стороні клієнта є те, що кількість необхідного JavaScript-коду має тенденцію зростати разом із ростом програми. Це стає особливо складним із додаванням нових бібліотек JavaScript, полізаповнень і коду сторонніх розробників, які конкурують за

обчислювальну потужність і часто мають бути оброблені перед відображенням вмісту сторінки. Додаток, створений за допомогою CSR, який покладається на великі JavaScript-сбірки, повинен реалізовувати метод значного поділ коду (code-splitting) та обов’язково відтерміновувати завантаження JavaScript (техніка lazy-loading) – «обслуговувати лише те, що вам потрібно, коли вам це потрібно». Для додатку з невеликою інтерактивністю або без неї рендеринг сервера може бути більш масштабованим рішенням цих проблем.

Для ситуацій, коли створюється односторінкова програма, визначення основних частин інтерфейсу користувача, спільного для більшості сторінок, означає, що можна застосувати техніку кешування частини програми. У поєднанні з Service Workers це може значно покращити продуктивність сайта під час повторних відвідувань.

3 АДАПТАЦІЯ БАЗОВИХ СТРАТЕГІЙ РЕНДЕРИНГУ ТА РОЗРОБКА ЗМІШАНИХ СТРАТЕГІЙ ПІД СУЧАСНІ ВИМОГИ

Сучасні вимоги до веб-додатків видозмінюються та потребують більш гнучких рішень в умовах конкуренції та вибагливості користувачів. Використання однієї стратегії для усіх сторінок суперечить меті створення продуктивного сайту та досягнення високої конверсії користувачів. Наприклад, деякі сайти наразі потребують впровадження блогу як частини додатка або ж поєднання статичних сторінок з тими, які мають надавати користувачу найбільш оновлені дані, не втрачаючи при цьому ефективності усіх видів сторінок.

У випадках, коли для додатку необхідно використати переважні можливості серверного рендерингу та залишити динамічність та гнучкість роботи за сторінкою на клієнті, а використання статичних HTML-сторінок неможливе через особливості системи або потоку даних в додатку, є доцільним адаптування відомих стратегій під поставлені задачі та розробка видозмінених стратегій рендерингу.

3.1 Поєднання рендерингу на сервері та клієнті

У випадках, коли веб-сайт потребує гарних показників SEO та високу інтерактивність на клієнті можна поєднати переваги рендерингу на стороні сервера та клієнта (SSR і CSR). Навігаційні запити, такі як запити сторінки або ж її перезавантаження, будуть оброблятися сервером, який будує та надсилає клієнту додаток у форматі готового HTML, а потім JavaScript і дані, що використовуються для відтворення, вбудовуються в кінцевий документ. При

коректній реалізації це також забезпечить швидку візуалізацію контенту (First Paint), як при серверному рендерингу.

Так, наприклад, типовий веб-додаток розроблений з React покладається на рендеринг на стороні клієнта використовуючи hydration – процес, при якому React перевіряє поточні DOM-елементи і наповнює їх даними або модифікує за допомогою відповідного JavaScript-коду. Замість обробки HTML для створення DOM клієнтська візуалізація використовує JavaScript для його створення. Мінімальний HTML-документ служить контейнером програми та містить лише посилання на JavaScript і CSS, необхідні для відтворення програми.

Основним недоліком такого підходу є те, що це може мати значний негативний вплив на FID, навіть якщо воно покращує First Paint. Сторінки можуть виглядати завантаженими та інтерактивними, але насправді будуть недоступні для взаємодії, доки не буде повністю виконано JS-код на стороні клієнта та не буде приєднано обробники подій (комп'ютерної миші, прокручування сторінки). Це може зайняти кілька секунд або навіть хвилин на мобільному пристрої.

Використовуючи поступове наповнення HTML дерева додатку, можна відстрочити модифікацію (й блокування інтерактивності) менш важливих частин сторінки. Таким чином, ми можемо зменшити обсяг JavaScript, який нам потрібно запитувати з серверу, щоб зробити сторінку інтерактивною, і оброблять DOM-елементи лише тоді, коли це буде потрібно користувачеві. Варіант поступового використання hydration також допомагає уникнути випадків, коли дерево DOM, побудоване сервером, руйнується, а потім негайно перебудовується.

Щоб усі компоненти на сторінці стали інтерактивними з використанням hydration, код React для цих компонентів має бути включений у сбірку, яка завантажується з серверу на клієнт. Наприклад, високоінтерактивні одно-сторінкові додатки, які в основному контролюються JavaScript-кодом, потребуватимуть одразу всього пакету сбірки. Однак для переважно

статичних веб-сайтів із кількома інтерактивними елементами на екрані може не знадобитися, щоб усі компоненти були активними негайно.

Послідовне наповнення вирішить цю проблему, дозволяючи нам активацію hydration для частини компонентів під час завантаження сторінки. Інші частини поступово будуються за потреби. На рисунку 3.1 зображено як працюватимуть серверні запити на сторінку та оброблятиметься JS-код при реалізації даної стратегії рендерингу.

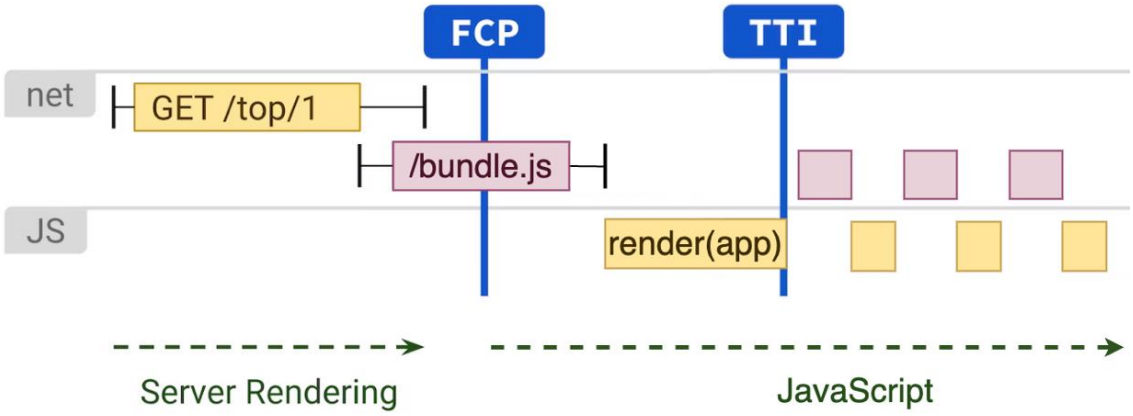


Рисунок 3.1 – Графічне зображення стратегії поступового наповнення

Для імплементації поступового наповнення можна використати одну з бібліотек: pool-attendant-preact або next-super-performance. Наприклад, бібліотека «next-super-performance» містить API під назвою «withHydration», який дозволяє позначати більш інтерактивні компоненти для активації hydration в першу чергу для них. На рисунку 3.1 зображено приклад запровадження описаної техніки в коді використовуючи JavaScript та React.

```
import { withHydration } from "next-super-performance";
import Component from "./component";

const HydratedComponent = withHydration(Component);

export default function SomePage() {
  return (
    <main>
      <Component column={1} />
      <HydratedComponent column={2} />
      <HydratedComponent column={3} />

      <Component column={1} />
      <Component column={2} />
      <Component column={3} />

      <Component column={1} />
      <Component column={2} />
      <Component column={3} />
    </main>
  );
}
```

Рисунок 3.2 – Приклад використання поступового наповнення з бібліотекою «next-super-performance»

Компонент «HydratedComponent» у стовпцях 2 і 3 буде доступним для взаємодії першим. Далі можна активувати інтерактивність решти компонентів на клієнті за допомогою API «hydrate()», який також включено до бібліотеки. На рисунку 3.2 зображено приклад коду активації решти компонентів.

```
import { hydrate } from "next-super-performance";
import Component from "./components/teaser";

hydrate([Component]);
```

Рисунок 3.3 – Приклад активації непріоритетних компонентів сторінки

Серед переваг такої стратегії рендерингу:

– завантаження тільки за потреби тих частини сторінки, які рідко використовуються користувачем;

– розділення коду (code-splitting) автоматично призводить до зменшення розміру пакета, що допомагає скоротити час між FCP і TTI.

3.2 Потокове надсилання на клієнт сторінок побудованих на сервері

Щоб скоротити час до доступності інтерактивності сторінки (метрика TTI), використовуючи рендеринг на стороні сервера можна впровадити техніку надсилання HTML-коду частинами, які браузер може поступово обробляти та відображати після отримання з серверу. Замість створення одного великого HTML-файлу, що містить необхідну розмітку для поточної сторінки, можна розділити його на менші частини. Потоки HTML-елементів дозволять передавати дані в об’єкті відповіді, що означитиме постійне надсилання даних на сторону клієнта. У той момент, коли браузер отримає фрагменти даних, він зможе почати його рендеринг.

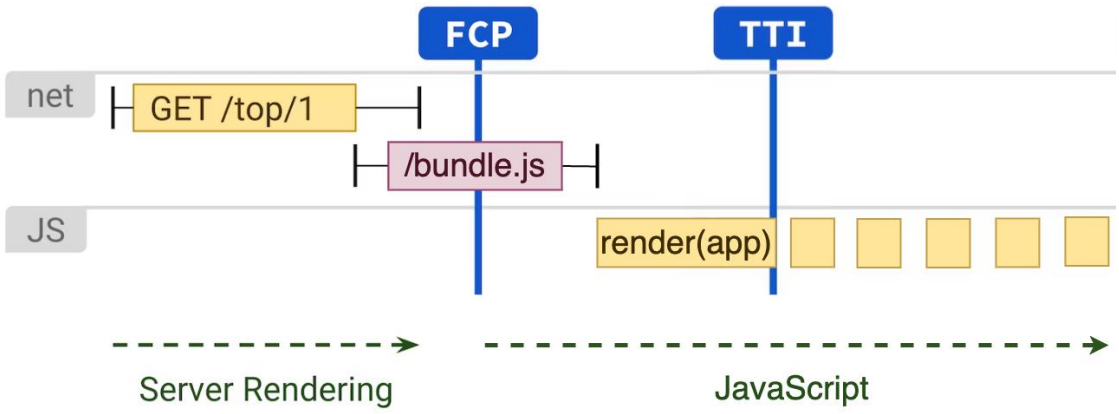


Рисунок 3.4 – Графічне зображення потокового рендерингу на стороні серверу

Наприклад, якщо додаток розроблюється із фреймворком React, розробник має вбудовану функцію «renderToNodeStream», яка дозволяє надсилати код додатку частинами в маленьких пакетах сбірки. При цьому подході браузер може почати будувати інтерфейс для користувача, коли він ще отримує дані, що забезпечує швидку візуалізацію сторінки (First Paint) та LCP, оскільки HTML швидше надходить до користувачів. Виклик React-методу «hydrate» на отриманих DOM-елементах приєднає відповідні обробники подій (event listeners), що зробить інтерфейс користувача інтерактивним.

Як приклад, розглядається веб-додаток зі списком даних на сторінці з кількістю елементів понад 1000. На рисунку 3.3 зображено клієнтську частину додатка, де будується елемент списку з елементами, які наповнюються даними.

```
import React from "react";
import dataList from "../data";

const App = () => {
  return dataList.map((listItem, i) => (
    <ul className="list">
      <li key={i}>{listItem.text}</li>
    </ul>
  ));
}

export default App;
```

Рисунок 3.5 – Клієнтська частина додатку

На стороні сервера використовується функція «renderNodeToStream». На рисунку 3.4 зображена частина файлу, де ініціюється базовий сервер додатка за допомогою бібліотеки express. Також створено частину HTML розмітки, яку клієнт отримає одразу. Цей HTML-фрагмент містить корисну інформацію, яку

програма має використовувати для правильного рендерингу сторінки, а саме назву документа, таблицю стилів та головний контент.

```
import React from "react";
import path from "path";
import express from "express";
import { renderToNodeStream } from "react-dom/server";
import App from "./src/App";

const app = express();

app.use("/client.js", (req, res) => res.redirect("/build/client.js"));
app.use((req, res, next) => {
  setTimeout(() => { next(); }, 500);
});

const BEFORE = `<!DOCTYPE html>
  <html>
    <head>
      <title>Home page</title>
      <link rel="stylesheet" href="/style.css" />
      <script type="module" defer src="/build/client.js"></script>
    </head>
    <body>
      <div id="approot">
`.replace(/s*/g, "");
```

Рисунок 3.6 – Перша частина серверного файлу

Далі на рисунку 3.5 зображено використання функції «renderToNodeStream». Головний компонент додатку App будується на сервері в потоці за допомогою «renderToNodeStream». Перша частина HTML надсилається в об’єкті відповіді разом із пакетами даних із компоненту «App».

У разі використання SSR для рендерингу компонента «App» користувачеві довелося б зачекати, доки програма не отримає всі дані, перш ніж вона зможе почати завантаження та обробку. Щоб пришвидшити це, renderToNodeStream дає змогу додатку почати завантажувати та обробляти інформацію і паралельно все ще отримувати фрагменти даних від компонента «App».

```
app.get("/", async (request, response) => {
  const stream = renderToNodeStream(<App />);

  stream.on("data", handleData = () => {
    stream.off("data", handleData);
    response.useChunkedEncodingByDefault = true;
    response.writeHead(200, {
      "content-type": "text/html",
      "content-transfer-encoding": "chunked",
      "x-content-type-options": "nosniff"
    });
    response.write(BEFORE);
    response.flushHeaders();
  });
  await new Promise((resolve, reject) => {
    stream.on("error", err => {
      stream.unpipe(response);
      reject(err);
    });
    stream.on("end", () => {
      response.write("</div></body></html>");
      response.end();
      resolve();
    });
    stream.pipe(
      response,
      { end: false }
    );
  });
});

app.use(express.static(path.resolve(__dirname, "src")));
app.use("/build", express.static(path.resolve(__dirname, "build")));
```

Рисунок 3.7 – Друга частина серверного файлу

Такий підхід випливає у певні переваги для сайту:

– оскільки розмітка досягає клієнта незабаром після початку рендерингу на сервері, показник Time to First Byte кращий, ніж для базового SSR;

– TTFB більш стабільний та передбачуваний на сторінках різного розміру;

– оскільки браузер може почати обробляти HTML, як тільки його отримує, показники FP і FCP мають кращий результат;

– потокове передавання добре реагує на зворотний тиск мережі або перевантаження та може призвести до адаптації веб-сайтів навіть за складних умов;

– сканери пошукової системи можуть зчитувати потокову відповідь, тому сторінка доступна для SEO.

4 АПРОБАЦІЯ РОЗРОБЛЕНИХ СТРАТЕГІЙ РЕНДЕРИНГУ

Для демонстрації застосування стратегії поступового рендерингу елементів сторінки веб-додатку було створено тестовий додаток з реалізацією To-do списку. Задача полягає в аналізі мережевих запитів клієнта на сервер та рендеринг «тимчасово» неінтерактивних компонентів додатку. Для першого завантаження сторінка має візуалізувати лише необхідні компоненти To-do списка, а в разі взаємодії користувача з будь-яким з них – додатково завантажити необхідні JavaScript-файли.

На рисунку 4.1 зображено початковий стан додатку разом з першими запитами JS-файлів, які включають сбірки файлів застосунку та файли сторонніх бібліотек.

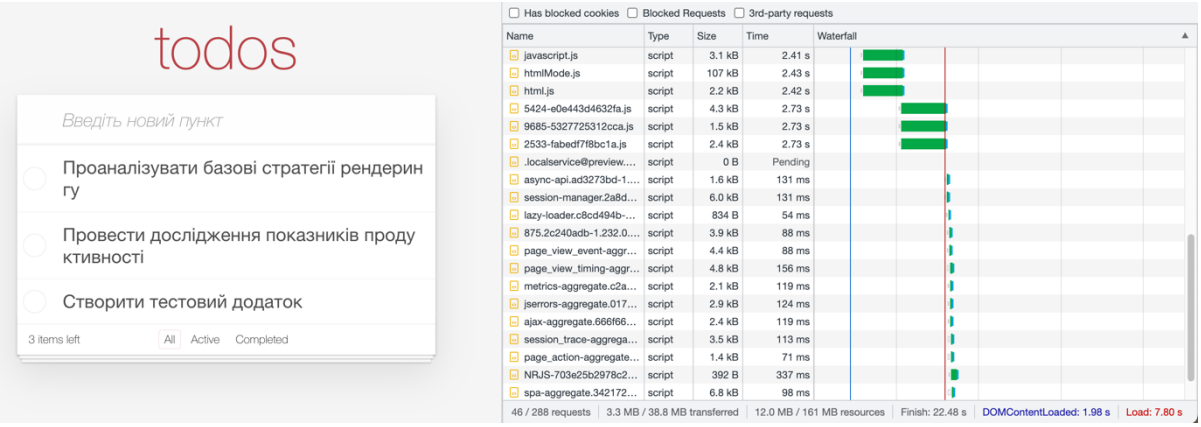


Рисунок 4.1 – Початковий стан застосунку з відображенням JS-запитів на сервер

При взаємодії з input елементом, за допомогою якого можна створити новий пункт списку, можна спостерігати як додаток виконує запити на сервер на певні компоненти додатку. Таким чином, клієнт отримує додаткові файли для наповнення та активації досі «статичних» компонентів, а користувач взаємодіє з елементами без жодних затримок.

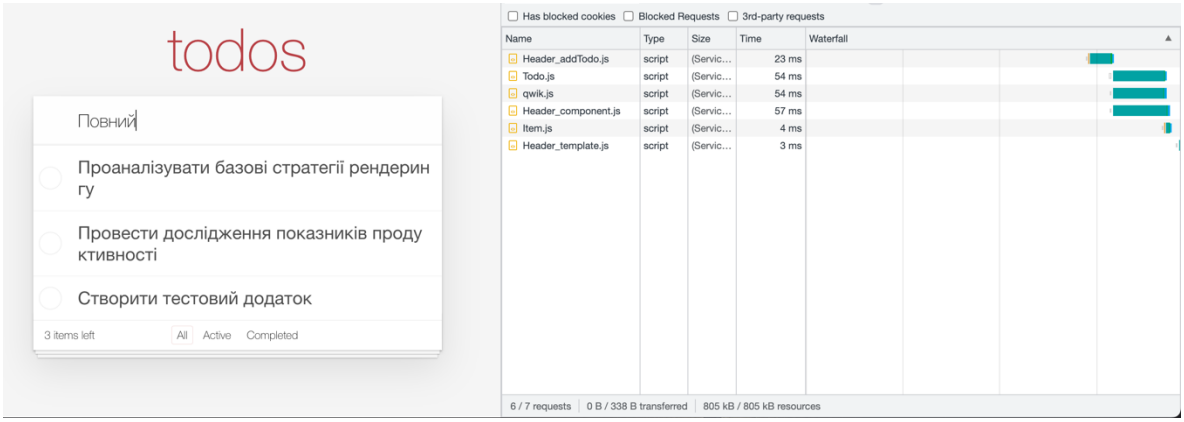


Рисунок 4.2 – Взаємодія з input елементом та запити на відповідні файли

У разі потреби відзначити пункт списку як виконаний, додаток знову зробить запит запит на необхідні для наповнення компоненти. Отримавши JS-файли з серверу, браузер обробить їх та елементи стануть доступними для взаємодії користувачем. Вкладка Network в браузері також демонструє короткий час виконання таких запитів, що значно оптимізує додаток – файли Item_edit.js та Item_remove.js завантажуються за 5 мілісекунд.

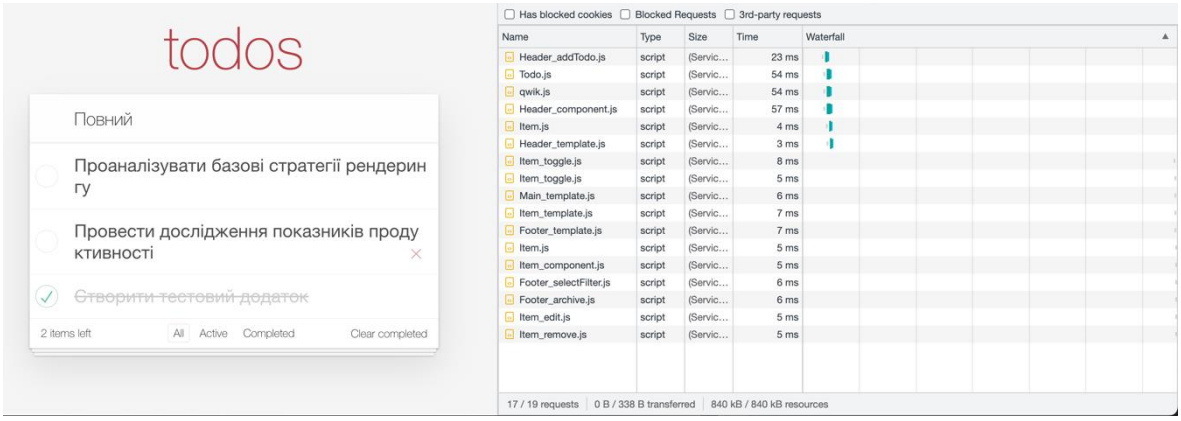


Рисунок 4.3 – Запити на компоненти для редагування та видалення елемента

Обираючи стратегії рендерингу завжди необхідно опиратися на бізнес-вимоги та можливі недоліки існуючого веб-додатку. Кожен підхід стане вигідним інструментом у розробці у разі чіткого розуміння цілей, пріоритетів, а також ресурсних можливостей проєкту. У таблиці 4.1 порівнюються розглянуті стратегії рендерингу з наведенням їх особливостей, переваг та недоліків.

Таблиця 4.1 – Порівняння стратегій рендерингу

Назва / характеристика	Client-Side Rendering	Server-Side Rendering	Static Site Generation	Поступовий клієнт. рендеринг	Потоковий серверний рендеринг
Короткий опис	SPA додаток – увесь процес будування додатку відбувається на стороні клієнта	При запиті додаток генерує сторінку на сервері на надсилає її на клієнт	Будування HTML-файлу під кожну сторінку сайту заздалегідь	Будування HTML на сервері та поступова обробка JS-коду на клієнті	Сервер. рендеринг HTML та надсилання на клієнт частинами, наповнення на клієнті
Рендеринг	DOM та JS	Динамічний HTML	Статичний HTML	Динам. HTML, JS	Динамічний HTML
Роль серверу у процесі	Надсилає статичний HTML	Виконує будування сторінки повністю	Надсилає статичний HTML згенерованої сторінки	Будує HTML сторінки, поступово надсилає частини усього JS-коду	Будує HTML сторінки маленькими частинами

Продовження таблиці 4.1

Назва / характеристика	Client-Side Rendering	Server-Side Rendering	Static Site Generation	Поступовий клієнт. рендеринг	Потоковий серверний рендеринг
Переваги	– легка та найбільш поширена у розробці – швидкий TTFB	– мізерна затримка між FCP і TTI – постійно «свіжі» дані з серверу	– найшвидший FP, FCP, LCP – швидкий TTFB	– мала затримка між FCP і TTI (при code-splitting) – пріоритетність компонентів	– TTFB кращий, ніж при SSR – добре працює в умовах тиску мережі та перевантаження
Недоліки	– повільний TTI – відсутня підтримка SEO	– повільний TTFB	– статичний HTML має бути створений для кожного URL – кількість збережених HTML файлів	– порівняно складна у імплементації – високий TTI для більшості елементів сторінки	– порівняно складна у імплементації

ВИСНОВКИ

У результаті виконання магістерської роботи виконано дослідження показників продуктивності веб-додатків та стратегій рендерингу як інструментальних засобів розробки веб-застосунків в рамках ІТ-проєктів.

Проведено аналіз інструментів frontend-розробки і методів оптимізації веб-сайту, показників продуктивності, видів рендерингу веб-сторінок. Було описано показники продуктивності і стратегій рендерингу з виявленням переваг та недоліків, порівняно особливості стратегій рендерингу.

У результаті проведеного аналізу було адаптовано існуючі стратегії рендерингу під сучасні вимоги ринку та розроблено змішані стратегії рендерингу. Аналіз та розроблені стратегії можуть бути використані при планування та розробці нових або існуючих веб-додатків, зможуть виявити слабкі місця сайту, покращити продуктивність, а також збільшити конверсію нових клієнтів бізнесу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура і правила оформлювання. – Чинний від 22.06.2015. – Київ: ДП «УкрНДНЦ», 2016. – 31 с.

2. ДСТУ 8302:2015. Інформація та документація. Бібліографічні посилання. Загальні положення та правила складаня. – Чинний від 04.03.2016. – Київ: ДП «УкрНДНЦ», 2016. – 20 с.

3. Методичні вказівки щодо розробки та оформлення магістерської атестаційної роботи за спеціальністю 122 Комп'ютерні науки (освітня програма «Управління проєктами в галузі інформаційних технологій» освітньо-кваліфікаційного рівня «магістр» / Упоряд.: Петров К.Е., Левикін В.М., Чалий С.Ф., Євланов М.В., Саєнко В.І., Міхнов Д.К., Міхнова А.В., Чала О.В. – Харків: ХНУРЕ, 2019. – 30 с.

4. How COVID-19 has pushed companies over the technology tipping point – and transformed business forever (5 жовтня 2020). URL: https://www.mckinsey.com/capabilities/strategy-and-corporate-finance/our-insights/how-covid-19-has-pushed-companies-over-the-technology-tipping-point-and-transformed-business-forever.

5. How the BBC builds websites that scale. URL: https://www.creativebloq.com/features/how-the-bbc-builds-websites-that-scale.

6. Mobile Internet traffic (20 грудня 2021). URL: https://explodingtopics.com/blog/mobile-internet-traffic.

7. Page Speed SEO – Cognitive SEO Blog. URL: https://cognitiveseo.com/blog/22865/page-speed-seo.

8. Driving user growth with performance improvements. URL: https://medium.com/pinterest-engineering/driving-user-growth-with-performance-improvements-cfc50dafadd7#.wwimdmkpp.

9. State of Online Retail Performance (Spring 2017) – Akamai and SOASTA. URL: https://s3.amazonaws.com/sofist-marketing/State+of+Online+Retail+Performance+Spring+2017+Akamai+and+SOASTA+2017.pdf

10. Vitals business impact. URL: https://web.dev/vitals-business-impact/

11. Process management – strategy, organization. URL: https://www.referenceforbusiness.com/management/Or-Pr/Process-Management.html

12. Core Web Vitals. URL: https://web.dev/vitals/

13. Optimize LCP. URL: https://web.dev/optimize-lcp