ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Вирішення проблеми часткового оновлення вмісту веб-сторінки шляхом оптимізації параметрів SPA»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Іван ХЛИСТА

Виконав: здобувач вищої освіти групи ПДМ-62
Іван ХЛИСТА

Керівник: Андрій БОНДАРЧУК
д.т.н., професор

Рецензент: _____
науковий ступінь,
вчене звання

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Хлисті Іванові Андрійовичу

1. Тема кваліфікаційної роботи: «Вирішення проблеми часткового оновлення вмісту веб-сторінки шляхом оптимізації параметрів SPA»

керівник кваліфікаційної роботи Андрій БОНДАРЧУК д.т.н., професор,

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «19» жовтня 2023 р. № 145.

2. Строк подання кваліфікаційної роботи «29» грудня 2023 р.

3. Вихідні дані до кваліфікаційної роботи:

1. Науково-технічна література, веб-розробка, оптимізація продуктивності.
2. Оновлення вмісту сторінки односторінкових веб-застосунків, параметри, що впливають на продуктивність SPA.
3. Користувацький досвід, надійність, стабільність клієнтської частин

4. Бібліотека React.js та фреймворк Angular.

5. Метрики Core Web Vitals сервісу PageSpeed Insights.

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

1. Порівняльний аналіз оптимізаційних технік оновлення вмісту клієнтської частини односторінкових веб-додатків.

2. Алгоритм визначення та оптимізації параметрів продуктивності односторінкового додатку на основі математичних моделей Нельсона, станів Маркова, стандарту APDEX-у.

3. Оптимізовані додатки на основі React.js та Angular для визначення ефективності часткового оновлення вмісту після застосування алгоритму.

4. Оцінка отриманих метрик продуктивності PageSpeed Insights додатків до оптимізації та після.

5. Перелік ілюстративного матеріалу: презентація та програмний код додатків

6. Дата видачі завдання «19» жовтня 2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	19.10.-05.11 23	
2	Дослідження особливостей функціонування процесу оновлення вмісту веб сторінки в односторінкових веб-додатках	06.11.-12.11.23	
3	Аналіз методів впливу на швидкість та якість оновлення вмісту веб-сторінки	13.11.-19.11.23	
4	Розробка алгоритму оптимізації параметрів впливу на продуктивність оновлення веб-сторінки	20.11.-26.11.23	
5	Проведення тестування показників продуктивності в додатках React та Angular на основі Core Web Vitals	27.11.-03.12.23	
6	Оформлення роботи та розробка демонстраційних матеріалів	04.12.-10.12.23	
7	Здача роботи	21.12.-29.12.23	

Здобувач вищої освіти _____ Іван ХЛИСТА

Керівник
кваліфікаційної роботи _____ Андрій БОНДАРЧУК

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 68 стор., 6 табл., 22 рис., 47 джерел.

Мета роботи – покращення користувацького досвіду під час взаємодії з веб-додатками шляхом підвищення їх продуктивності.

Об'єкт дослідження – процес часткового оновлення вмісту веб-сторінки

Предмет дослідження – методи, що впливають на часткове оновлення клієнтської частини веб-сторінки

Короткий зміст роботи:

Дане дослідження присвячене проблемі незадовільного користувацького досвіду, котрий виникає під час оновлення сторінки в процесі життєвого циклу додатку внаслідок взаємодії користувача з нею в браузері. З метою визначення причин та можливостей поліпшення процесу оновлення вмісту сторінки в роботі проаналізовані особливості функціонування односторінкових веб-додатків, в основі яких використовується найсучасніша на сьогодні техніка часткового оновлення вмісту сторінки. Встановлено, що причини проблеми криються в недостатній оптимізації параметрів, що відповідають за продуктивність додатку в цілому.Представлено експериментальну систему оптимізації на основі розробленого алгоритму із застосуванням математичних моделей Нельсона, станів Маркова, стандарту APDEX-у. Наведена оцінка ефективності розробленого підходу на основі стороннього сервісу PageSpeed Insights шляхом вимірювання метрик Core Web Vitals та порівняння показників продуктивності додатків на основі бібліотеки React.js та фреймворку Angular до застосування системи та після, включно із розгорнутим поясненням

КЛЮЧОВІ СЛОВА: ОНОВЛЕННЯ ВМІСТУ ВЕБ-СТОРІНКИ, ВИД ВЕБ-СТОРІНКИ, ШВИДКІСТЬ ЗАВАНТАЖЕННЯ, МЕТОДИ ОПТИМІЗАЦІЇ ПРОДУКТИВНОСТІ.

ABSTRACT

Text part of the master's qualification work: 68 pages, 22 pictures, 6 tables, 47 sources.
The purpose of the work: improvement of user experience during interaction with web applications by enhancing their productivity.
Object of research: partial content update process on a web page.
Subject of research: methods influencing the partial update of the web page client.

Summary of the work:

This research is dedicated to the issue of unsatisfactory user experience that arises during the page update process in the application's lifecycle due to user interaction in the browser. In order to identify the causes and possibilities for improving the page content update process, the functioning features of single-page web applications are analyzed, which are based on the modern technique of partial content updating.

The roots of the problem lie in the insufficient optimization of parameters responsible for the overall application performance. The necessity of a comprehensive approach to improve performance metrics is justified, a comparative analysis of existing methods for optimizing the performance of the web applications clients is conducted.

An experimental optimization system is presented based on a developed algorithm using mathematical models such as Nelson, Markov states, and the APDEX-y standard. The effectiveness of the developed approach is evaluated using the PageSpeed Insights third-party service by measuring Core Web Vitals metrics and comparing performance indicators of applications based on the React.js library and Angular framework before and after the application of the system, including detailed explanations.

KEYWORDS: WEB PAGE CONTENT UPDATE, WEB PAGE VIEW, PAGE LOADING SPEED, PERFORMANCE OPTIMIZATION METHODS, REACT, ANGULAR, CORE WEB VITALS METRICS

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AJAX	Asynchronous JavaScript and XML	Асинхронний JavaScript та XML
APDEX	Application Performance Index	Індекс продуктивності додатку
CDN	Content Delivery Network	Мережа доправлення контенту
CMS	Content Management System	Система управління контентом
CSS	Cascading Style Sheets	Каскадні таблиці стилів
DOM	Document Object Model	Об'єктна модель документа
FID	First Input Delay	Затримка першого введення
FMP	First Meaningful Paint	Перше значуще зображення
HTML	Hypertext Markup Language	Мова розмітки гіпертексту
JS	JavaScript	JavaScript
JSON	Javascript Object Notation	Нотація об'єктів JavaScript
PLT	Page Loading Time	Час завантаження сторінки
RIA	Rich Internet Application	Багатофункціональний Інтернет-додаток
SI	Speed Index	Індекс швидкості
SPA	Single Page Application	Односторінковий веб-додаток
TTFB	Time To First Byte	Час до першого байта
UI	User Interface	Користувацький інтерфейс
W3C	World Wide Web Consortium	Всесвітній Консорціум Світової Павутини

ВСТУП

Оптимізація — один із ключових та найбільш суттєвих елементів розробки та подальшого обслуговування будь-якої програми, а тим паче, коли мова йде про веб-додатки. Продуктивність, оперативність реагування, споживання ресурсів браузера, навантаження, загальний час завантаження, передбачуваність, плавне відображення — ці та багато інших аспектів стосуються процесу поліпшення функціонування веб-сторінки. У сучасному світі такі додатки стали значно більш орієнтованими на користувача ніж були колись раніше, а отже значно складнішими та вимогливішими як з точки зору розробки, так і з боку очікувань відвідувачів сайтів.

Одним із аспектів такої складності та водночас потужності, що застосовується в клієнтській розробці, є техніка часткового оновлення змісту сторінки додатку. Ця техніка сьогодні використовується майже в усіх технологіях клієнтської частини веб-розробки, та сприймається користувачами як щось абсолютно стандартне та цілком очікуване. Проте проблема цієї техніки полягає в тому, що вона потребує свідомого управління та постійного коригування, оскільки будучи потужним інструментом і даючи широкі можливості для якісного відображення користувацького інтерфейсу, вона несе з собою додаткові ризики для безперешкодного функціонування збірки. Тому просто послуговуватись технікою не означає гарантувати якісний функціонал користувачеві.

Якщо очікування користувача не задовольнити, то в подальшому він може відчувати незручності в користуванні та врешті-решт відмовитись від відвідування веб-сайту, який не відповідає його очікуванням. Покращення продуктивності може значною мірою вплинути на користувацький досвід, збагачуючи його та як наслідок сприяючи збільшенню відвідуваності електронного ресурсу.

Актуальність даного дослідження також підтверджується збільшенням та постійним оновленням технологій, які використовуються для клієнтської веб-розробки, а також із зростанням кількості активних веб-сайтів щороку [1].

Тому ефективним рішенням проблеми часткового оновлення вмісту веб-сторінки є аналіз особливостей розробки веб-додатку в залежності від конкретного інструменту та порівняння показників ефективності функціонування такого додатку до проведення оптимізації та після.

Мета роботи: покращення користувацького досвіду шляхом вирішення проблеми часткового оновлення вмісту веб-сторінки за рахунок оптимізації її параметрів.

Для досягнення цієї мети в роботі необхідно вирішити такі **завдання:**

1. Проаналізувати наукові праці з досліджуваної проблеми і обґрунтувати застосування конкретних технік оптимізації для поліпшення користувацького досвіду.
2. Дослідити роль SPA-додатків у порівнянні із іншими видами сайтів, зокрема із традиційними мультисторінковими сайтами.
3. Вивчити принципи функціонування техніки часткового оновлення вмісту веб-сторінки.
4. Розглянути особливості оптимізації продуктивності SPA на прикладі найбільш поширених фреймворків для клієнтської веб розробки Angular та React.
5. Розробити алгоритм визначення потенційних слабких місць в роботі клієнтської частини веб-додатку та запропонувати механізм їх поліпшення на основі підвищення продуктивності роботи застосунку.
6. Провести тестування показників продуктивності на основі загальновизнаних метрик Web Vitals до та після внесення змін для перевірки їх ефективності.

Виходячи з цього, **об’єктом дослідження** є процес часткового оновлення вмісту веб-сторінки, а **предметом дослідження** — методи, що впливають на часткове оновлення клієнтської частини сторінки

Методи дослідження: порівняльний, аналізу, виміру та експерименту

Наукова новизна одержаних результатів: розроблено рекомендації щодо поліпшення користувацького досвіду клієнтської частини веб-додатку на основі технік оптимізації процесу оновлення змін виду веб-сторінки та її продуктивності.

Практичний результат: результати магістерської роботи та її основні пункти можуть застосовуватися як безпосередньо для розробки програмного забезпечення клієнтської частини веб-додатків, так і для вивчення навчальних дисциплін пов’язаних із нею шляхом впровадження їх у навчальний процес вищих навчальних закладів.

Апробація результатів дослідження:

1. Бондарчук А.П., Хлиста І.А. Вирішення проблеми часткового оновлення вмісту веб-сторінки шляхом оптимізації параметрів SPA // V Всеукраїнська науково-практична конференція студентів, аспірантів та молодих вчених “Комп’ютерне моделювання та інформаційні технології” – Львів: НЛТУ, 2023 [2]. С. 286 - 291.
2. Bondarchuk A. Khlysta I. Mitigating partial content update issues on web pages through SPA parameter optimization. State University of Information and Telecommunication Technologies. “Telecommunication and Information Technologies” 2023. [3].

1 АНАЛІЗ ТЕХНОЛОГІЙ ОНОВЛЕННЯ ВМІСТУ ВЕБ-СТОРІНКИ

Під техніками часткового оновлення вмісту веб-сторінки науковці розуміють певний набір засобів реагування на зміни виду (view) статичної веб-сторінки під час перегляду та взаємодії із нею користувача. Процес впровадження засобів, які б могли постійно підтримувати стандартне функціонування сторінки так, щоб з одного боку максимально ефективно відповідати на запити користувача завдяки кращій продуктивності, а з іншого боку дозволяти полегшити технічне обслуговування коду та зберігати відповідний стандарт якості ніколи не досягне кінця [4].

Складність та багатошаровість логіки веб-сайтів та інформаційних платформ постійно зростають. Саме через це звичайні статичні веб-сайти чи їх альтернативні динамічні рішення, зокрема ті, що базуються на CMS не в змозі більше задовольнити потреби користувачів.

Ключову роль для сучасного користувача тепер відіграють час, якість та швидкість взаємодії з клієнтом, які в світі цифрових технологій дедалі більше наближаються до мілі-, якщо не мікро показників.

І хоча найперший поштовх до розвитку технік часткового оновлення вмісту веб-сторінки дали роботи Тіма Бернера-Лі на початку 90-их років минулого століття, все ж для їх більш повноцінного аналізу варто зупинитись на основних аспектах, категоріях, структурах технік, їх еволюції та ролі в щоденній веб-розробці [5, с.1].

Для цього розглянемо спочатку як з питанням змін та оновлення виду сторінки справлялись до появи підходу SPA який приніс із собою механізм часткового оновлення сторінки.

1.1. Порівняння змін виду сторінки в різних підходах веб-розробки

Із започаткуванням серверних мов програмування та власне з винаходом веб-серверів, розробники відкрили для себе можливість створювати серверно-орієнтовані веб-додатки.

Серверно-орієнтовані веб додатки або як їх прийнято називати у вітчизняній науковій літературі — *традиційні веб-додатки* — це додатки, що повністю функціонують в браузері та не вимагають встановлення на клієнті. Оновлення виду традиційного веб-додатку відбувається на веб-сервері і лише потім плавно передається клієнту, оскільки в основі принципу функціонування додатку такого типу лежить передача цілої відображуваної в браузері сторінки як відповідь у формі HTTP запиту на будь-яку спробу користувача взаємодіяти із додатком [5].

До прикладу розгляньмо звичайний прототип веб-сторінки краєзнавчого музею міста. Користувачеві потрібно умовно перейти із головної сторінки на сторінку історії заснування музею. За сьогоднішніми мірками, така навігація користувача по сайту не зайняла б більше ніж декілька мілісекунд.

У традиційних веб-додатках кожна дія користувача як результат спричиняла б цілковите оновлення сторінки браузера, у той час як змінена або ж нова сторінка паралельно створювалась б веб-сервером заново та відправлялась б у відповідь HTTP-запитом. Як результат — весь вміст старої сторінки замінювався б на нову.

Як бачимо на рисунку 1.1., традиційні веб-додатки використовують серверно-орієнтований підхід, за якого все навантаження покладається виключно на сервер, у той час, як клієнт виконує примітивну роль візуалізації інтерфейсу користувача UI.

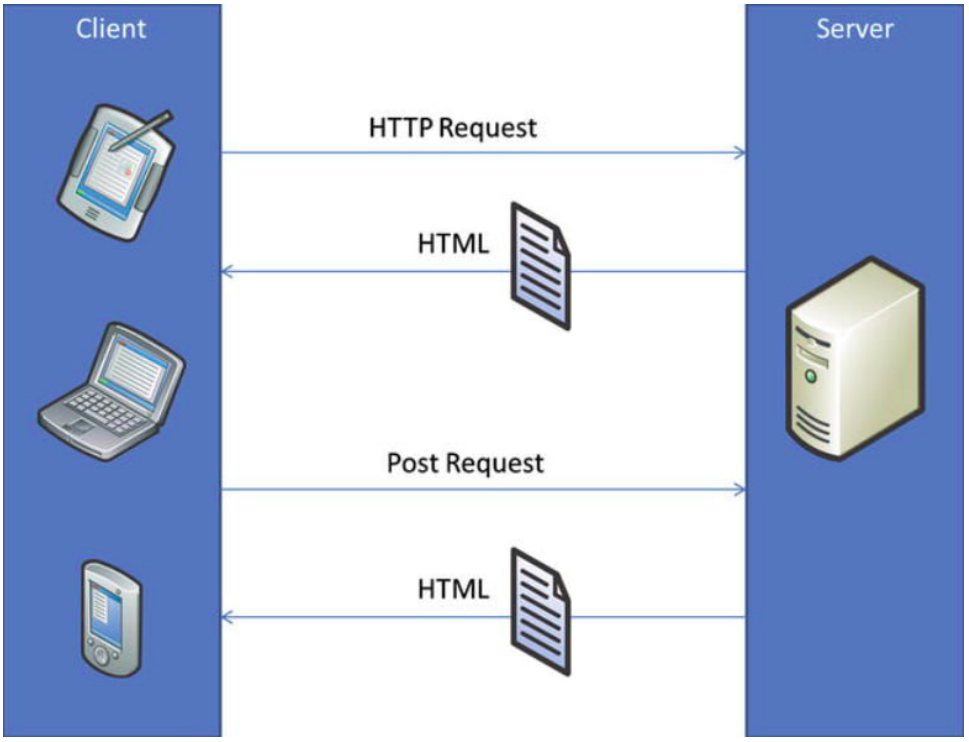


Рис. 1.1 Життєвий цикл сторінки у традиційному веб-додатку

Повернемося до нашого прикладу із використанням примітивної навігації задля оновлення вмісту веб-сторінки. Як наслідок обробка таких дій користувача відповідно до потенційного навантаження зазначеного в рисунку 1.1. могла б вимагати, по-перше, значних ресурсів серверу, по-друге, призвести до довгого очікування поки оновлена сторінка повернеться назад та буде заново відображена в браузері.

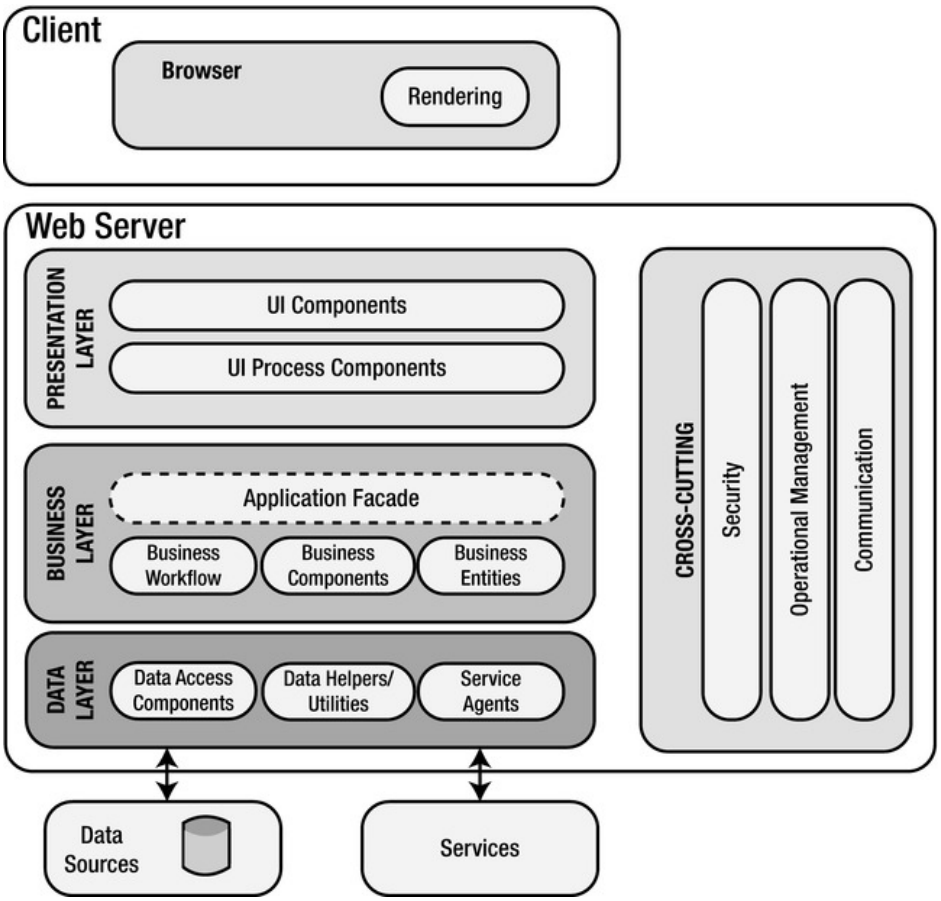


Рис. 1.2.Архітектура традиційного веб-додатку

З іншого боку, якщо розглянути рисунок зрізу архітектури такого веб-додатку (рис. 1.2.), то крім непропорційного співвідношення між клієнтською та серверною частинами додатку, не зовсім зрозумілими є особливості збереження стану та управління даними, оскільки це створює нові труднощі, а саме необхідність надсилати запити на сервер завжди, коли клієнту потрібні нові дані: навіть тоді, коли б ці дані можна було б зберегти або безпосередньо в браузері, або на стороні клієнту, а це — знову ж таки новий запит, а отже використання значних ресурсів, збільшення часу очікування для користувача та як наслідок низька адаптивність та відзивчивість традиційного веб-додатку.

На противагу традиційним веб-додаткам, *натівні додатки* не мають такої жорсткої прив’язаності до серверу і можуть не лише зберігати свій локальний стан, а й постійно оперувати ним завдяки використанню локальних баз даних чи файлів ОС. Це значно полегшує розробку інтерфейсу користувача та покращує

адаптивність сайту, адже в першу чергу використовуються локальні ресурси, а не ресурси віддалених серверів, а крім цього з'являється можливість запропонувати користувачеві доступність веб-додатку навіть попри відсутність стабільного інтернет з'єднання [6, 7].

Проте нативні додатки в порівнянні із традиційними мають додаткові обмеження у вигляді співвідношення із платформами під які вони розробляються.

Зважаючи на суттєві переваги, які пропонують нативні веб-додатки, на жаль, вони не вирішують проблеми кросплатформеності, яка в даному випадку залежить від ОС, версій драйверів та архітектури пристрою, а це значить десятки версій для сотень різновидів програмного середовища кожного потенційного користувача. Однак для швидшого та якіснішого прогресу світ потребував чогось суттєвішого [7].

Науковці Консорціуму Всесвітньої Мережі W3C, досліджуючи проблематику оновлення сторінки веб-додатку, дійшли до висновку, що застосування технології AJAX для більш рівномірного розподілу навантаження між клієнтом та веб-сервером дало б змогу користуватись одночасно перевагами як традиційних, так і нативних веб-додатків при цьому зменшуючи кількість необхідних запитів до віддаленого веб-серверу, а також поступово відображати певну змінену частину усієї раніше завантажуваної сторінки [6].

Оскільки технологія AJAX на час проведення дослідження не була новою, то важко стверджувати, що результати дослідників спричинили революцію в наукових колах, однак для розробників отримані результати стали каталізатором започаткування нової, гібридної концепції до оновлення веб-сторінки, втіленої в підході *RIA*.

Поняття RIA, як правило, тлумачиться як багатий на … інтернет додаток. Проте йому важко приписувати одне конкретне визначення, оскільки такого в наукових колах й досі не існує.

Так, група дослідників тематики RIA-додатків вважає, що під поняттям RIA варто розуміти широку множину можливостей взаємодії саме з інтерфейсом

користувача, яка включає в себе анімації, підтримку відеоматеріалів, функціонал drag and drop та врешті-решт 3D-ефекти [6].

Е. Скотт пов’язує “багатство” RIA-додатків саме із тягарем розробки, який із появою концепції більш-менш пропорційно розділився між клієнтом та сервером відповідно [7].

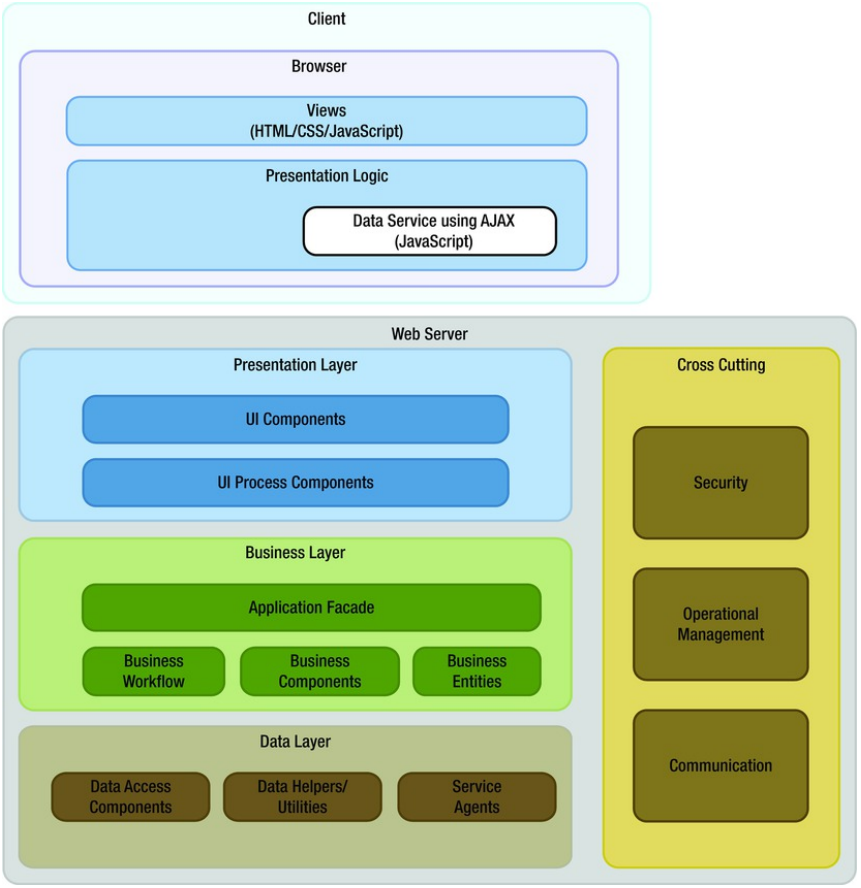


Рис 1.3. Архітектура RIA додатку

Дійсно, аналізуючи рис 1.3. можна констатувати, що додатки типу RIA виконують значною мірою “презентаційну” логіку на клієнті завдяки застосуванню технології AJAX. Також клієнт частково виконує бізнес логіку.

Однак дечого все що не вистачало. Підхід RIA все ще не повністю відмежовував перезавантаження сторінки при оновленні даних, отриманих із сервера та міг структурно складатись більше ніж із однієї сторінки, перешкоджаючи “повноцінному” частковому оновленню.

1.2. Аналіз ролі SPA у веб-розробці

Вважається, що точне визначення SPA як концепції розробки односторінкових веб-додатків в науці не існує так само як і визначення згаданого раніше RIA підходу, хоча існує безліч контраверсійних думок.

Найбільш прийнятним, на нашу думку, є позиція Месбаха та Ван Доєрсена, які під значенням SPA розуміють повноцінний веб-додаток, що має лише одну окрему сторінку, котра використовується ніби оболонка для всіх інших компонентів (сторінок) цього веб-додатку на основі JavaScript, CSS та HTML та складається із окремих підкомпонентів [9, с. 181–190].

До того ж вчені чітко вказують на унікальність такого додатку, адже він завантажує всі ресурси при першій ініціалізації та “довантажує” в міру взаємодії користувача [9, с. 181–190].

На противагу традиційній або так званій MPA *(Multi Page Application з англійської - багатосторінковий додаток)* розробці, яка відповідно до своєї назви взагалі не містить ні найменшого натяку на оновлення вмісту веб-сторінки, а тільки завантажує щоразу нові сторінки, SPA-додатки значною мірою не покладаються на взаємодію із веб-серверами, а здебільшого користуються перевагами, які пропонує розробка клієнтської частини із застосуванням технології асинхронного запитування даних AJAX.

Для порівняння SPA додатку та традиційного/нативного MPA додатку розглянемо схему життєвого циклу веб-сторінки, зображену на рисунку 1.4.

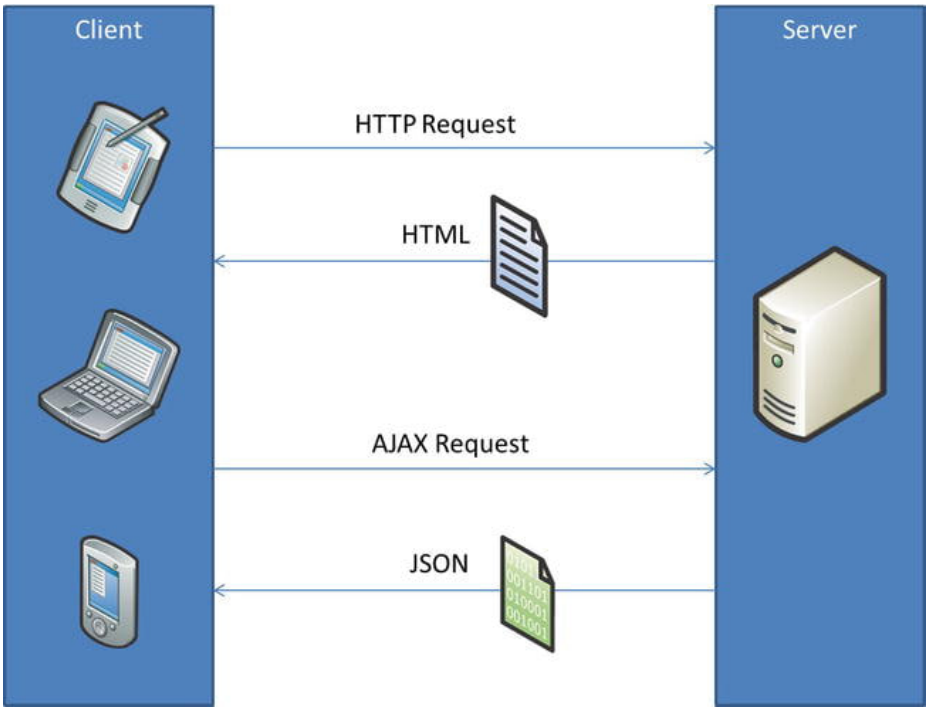


Рис 1. 4. Життєвий цикл SPA додатку

Найпершою зміною яку одразу можна помітити неозброєним оком на рисунку є характер запитів та формат відповіді на ці запити, які надсилає сервер. SPA широко послуговується технологією AJAX, яка асинхронно передає дані від клієнта до сервера без необхідності повністю оновлювати всю сторінку кожен раз заново та базується на принципах AMD *(з англійської Asynchronous Module Definition - асинхронне визначення модулів)* [6].

Оскільки більшість веб-сайтів сьогодні містить низку повторюваного матеріалу, будь то заголовки, описи, правові положення чи (в залежності від тематики) безпосередньо товари, здебільшого вони постійно повторюються або в одному конкретному розділі, або в цілому по всьому додатку. Уникнути цього контенту неможливо, але односторінкові додатки використовують це повторення завдяки асинхронності на свою користь.

До того ж передача даних здійснюється у форматі JSON, що сприяє прискоренню клієнт-серверної комунікації та підтримується багатьма мовами програмування. Як тільки дані надходять від серверу до клієнту, останній відображає зміни ніби динамічно перезаписуючи вид сторінки. У зв’язку з тим,

що користувач не залишає головну сторінку в такому випадку та не оновлює її, можна скористатись пам’яттю браузера для управління станом додатка, що є значним досягнення для SPA додатків [10].

Підсумовуючи, визначимо переваги та недоліки односторінкового додатка у порівнянні з його аналогами [11]:

1. SPA доступні для великої кількості різних пристроїв, тому, розробникам не слід перейматись створенням щоразу нового додатку для пристроїв із різними ОС, розмірами екрану, налаштуваннями.
2. Односторінкові додатки не потребують додаткових розширень для налагодження в веб-переглядачах.
3. SPA ефективно кешують дані в локальному сховищі.
4. Завдяки тому, що веб-сторінка одна,- легше будувати багатий UI.
5. Низькі витрати на підтримку в довгостроковому плані.
6. Важкі клієнтські фреймворки, які потрібно завантажувати на клієнт.
7. Практично повна відсутність SEO (*з англійської - Search Engine Optimization - оптимізація пошукових систем*) підтримки через те, що не всі пошукові системи підтримують рендеринг сторінки на стороні клієнта.
8. Одна помилка може призвести до розладнаності всього додатку.

1.3. Техніки розробки односторінкового додатку

Веб-сайт — це своєрідний будиночок, в який має завжди бути багато входів та виходів. Так само і розробка не повинна обмежуватись лише одним варіантом та розглядати принаймні декілька способів для визначення найбільш підходящого з точки зору завдань, цілей, поставленої задачі та ін. [12].

Розглянемо деякі техніки побудови односторінкового додатку та їх еволюцію в контексті механізму оновлення вмісту веб-додатку.

Прототипом усіх наявних сьогодні способів побудови односторінкового додатку стала техніка глибоких посилань, відома також під назвою техніка гіперпосилань або ж глибокого введення.

Глибоке посилання *(з англійської — deep linking)* — це спосіб використання посилань, які закріплюються за певною конкретною частиною веб-додатку та при натиску на них перенаправляють користувача до відповідного блоку змісту сторінки. Сторінка у такому випадку все ще суттєво покладається на серверну комунікацію (потребує місткого основного бандлу при ініціалізації), але користується перевагами звичайного html закріплення *(з англійської anchoring — закріплення/закидати якір)* [13].

Основними здобутками глибоких посилань стало створення легкої збірки, що ефективно перенаправляла користувача з однієї частини сторінки до іншої, викладаючи необхідний контент. Це стало можливим завдяки взаємодії із URL браузера та розвитком HTML5 History API [12].

Спеціальний шаблонізатор, або система вебшаблонів — це техніка розробки додатку, яка послуговується готовими макетами html та моделями даних, складеними різними мовами програмування для автоматичного генерування типових веб-сайтів. Результати функціонування такого підходу можна доволі часто зустріти серед новинних веб-ресурсів, блогів або ж поміж культурних порталів, як наприклад сторінка журналу TIME Magazine (https://time.com/), SONY Music (https://www.sonymusic.com/), VOGUE (https://vogue.ua), TED Blog (https://blog.ted.com).

Ядром цієї системи є двигун/процесор шаблону, що по суті поєднує макет html та відповідну модель даних в одну збірку, яка врешті-решт і є кінцевим веб-додатком [6].

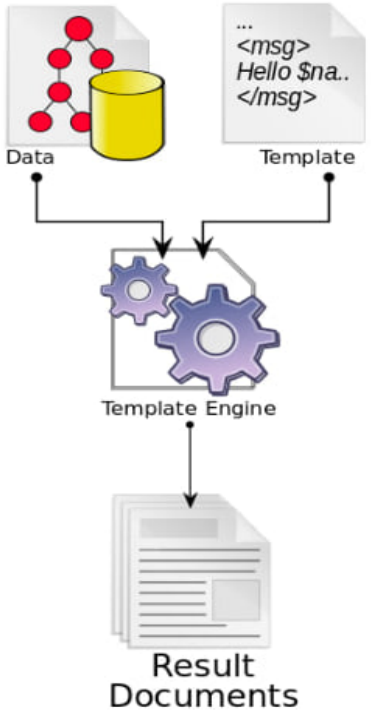


Рис. 1.5. Принцип роботи системи шаблонів

Завдяки розділенню логіки програмування від візуального представлення, техніка спрощує процес розробки, адже уможливлює перевикористовуваність схожих для декількох сайтів візуальних компонентів.

Її принципи функціонування нагадують роботу CMS, але варто відрізняти ці два поняття оскільки CMS лише використовує систему шаблонів для підтримки консистентності вигляду сайту та управління ним, надаючи користувачеві більше простору дій без необхідності занурюватись у глибини аспектів програмування, зокрема й кросплатформеність [6].

Попри вказані переваги, система шаблонів не є гнучким вирішенням проблеми часткового оновлення вмісту веб-сторінки, оскільки великі збірки файлів часто збільшують споживання ресурсів, потрібних на їх обслуговування, а підтримка оновлення шаблонів часозатратний процес, що потребує відповідного досвіду.

Для вирішення цієї комплексної проблеми необхідно було б запропонувати сталу модель поведінки, яка б містила уніфіковані методи для обслуговування

додатку та водночас була б широко доступним засобом. Ним стали сучасні JavaScript бібліотеки.

Бібліотеки в програмування — це збірки, колекції неперемінливих моделей поведінки в розумінні мови програмування, що використовуються для розробки програмного забезпечення. Вони можуть включати конфігураційні дані, документацію, допоміжні дані, попередньо визначений програмний код, а також класи, значення та специфікацію типів [14].

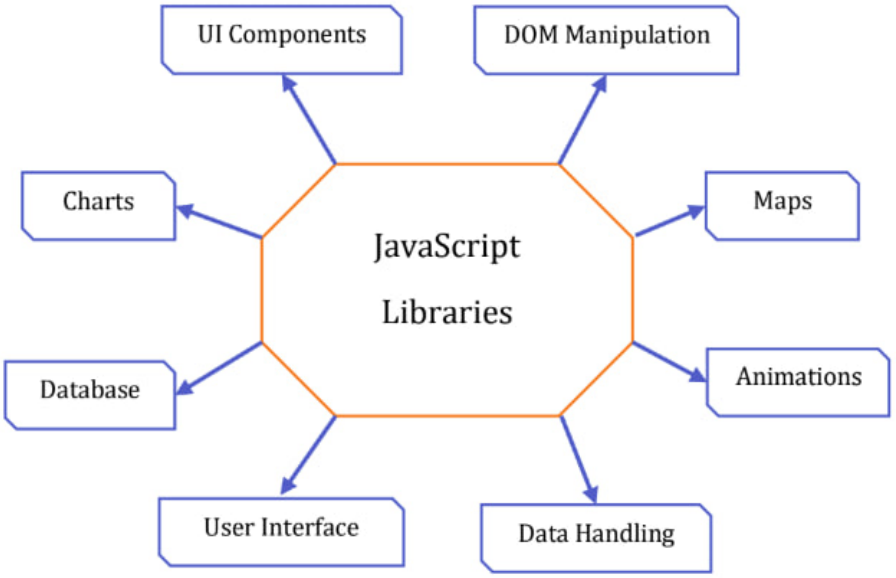


Рис. 1.6. Схематичне зображення типів JS-бібліотек

Як відображено на рисунку, такі задачі можна відповідно згрупувати по категоріях в залежності від їх призначення та функціональності. Бібліотеки, завдяки яким розробляються додатки типу SPA, належать до категорії User Interface. Існує безліч UI-бібліотек, що використовуються при проектуванні додатку, однак більш детально ми розглянемо їх застосування на практиці під час імплементації алгоритму оптимізації продуктивності веб-додатку.

Тепер виділимо ключові особливості JavaScript UI-бібліотек в контексті розробки односторінкових додатків.

Таблиця 1.1

Переваги та недоліки розробки методом бібліотек JS

Переваги	**Недоліки**
1. Прискорення розробки; Бібліотека як правило надає конкретний, готовий функціонал для вирішення певної проблеми.	1. Складність налаштування та необхідність завантаження значної кількості підмодулів та плагінів
2. Кросплатформеність; Кінцевий продукт розробки функціонує одночасно на багатьох платформах без ручного налаштування під кожний конкретний випадок.	2. Велика тривалість ініціалізації кінцевої збірки в браузері і як наслідок падіння користувацького досвіду
3. Модульність; Кінцевий код легко розподіляється на модулі, кожен з яких має свою зону відповідальності, що полегшує подальше обслуговування.	3. SPA, розроблений на основі JavaScript UI-бібліотеки має стандартний рівень захищеності, а це призводить до підвищеної вразливості, проблем із збереженням даних
4. Стандартизація; Програмний код не є хаотичним, а відповідає чітким стандартам особливо в контексті командної роботи.	
5. Зменшення навантаження; Бібліотека містить готові додаткові рішення, які можна використовувати автоматично, наприклад вбудовані рішення для захисту від вразливостей.	

У порівнянні із шаблонізаторами, глибокими посиланнями чи навіть CMS-системами стає очевидним, що для якісної, швидкої та передбачуваної роботи веб-сторінки важливо скористатись перевіреним та більш-менш використовуваним інструментом, яким власне у дослідженні й виступають збірки JavaScript UI-бібліотек.

Загалом, у цьому розділі йшлося про підходи веб-розробки, що застосовують техніку оновлення вмісту веб-сторінки для відображення кінцевого виду документу. Детально розтлумачується роль односторінкового веб-застосунку та досліджуються інструменти розробки, що мають більше практичне значення для проведення емпіричного експерименту в ході подальшого вивчення питання. Наголошується на особливій ролі JavaScript UI-бібліотек та перелічуються позитивні та негативні сторони їх використання.

2 МЕТОДИ ОПТИМІЗАЦІЇ ПАРАМЕТРІВ ПРОДУКТИВНОСТІ ДОДАТКУ

2.1. Загальні параметри веб-сторінки, що впливають на продуктивність

Параметри оптимізації продуктивності веб-додатку — це загальна сукупність усіх показників, факторів, процесів, що відповідають за роботу застосунку та можуть бути покращені в силу своїх програмних характеристик [18, 19].

Сучасні проекти створюються беручи до уваги практику CI/CD (з англ. Continuous Integration/Continuous Delivery — постійної інтеграції та забезпечення), що передбачає потребує аналізу ключових показників їх конкурентоспроможності, ефективності та продуктивності на кожному етапі існування проекту.

Безпосередньо від своєчасного аналізу, виявлення та усунення потенційних вразливостей продуктивності залежать такі значущі для кінцевих власників фактори, що впливають на популярність, привабливість та прибутковість веб-сайту [20]:

1. Кількість користувачів та трафік: Клієнти схильні відмовлятися від веб-сторінок, які завантажуються повільно та змушують довго очікувати, відповідно кількість залучених користувачів або зростає, або зменшується.
2. Користувацький досвід: Продуктивність сторінки впливає на її зручність, а остання формує загальний користувацький досвід.
3. Дохід: Онлайн-дохід безпосередньо пов'язаний з продуктивністю ключових сторінок і транзакцій для сайтів електронної комерції.
4. Рейтинг у пошукових системах: Показники оптимізації сторінки є одним з критеріїв рейтингу, який формують пошукові системи та використовують при відображенні найбільш підходящих результатів.
5. Багатоканальність: Мобільні пристрої або планшети легше отримують доступ до оптимізованих сторінок.

За кожен із визначених факторів відповідає один або декілька оптимізаційних параметрів, які найкраще об’єднати в групи, адже таким чином набагато легше проаналізувати методи та підходи до оптимізації такої сукупності досліджуваних параметрів.

У працях [18, 20] наводяться чотири основні категорії оптимізації продуктивності та їх складові компоненти, а саме :

- Розробка дизайну веб-додатку (архітектура побудови застосунку, програмний код).
- Управління мережевою інфраструктурою.
- Організація активів.
- Загальна оптимізація процесів.

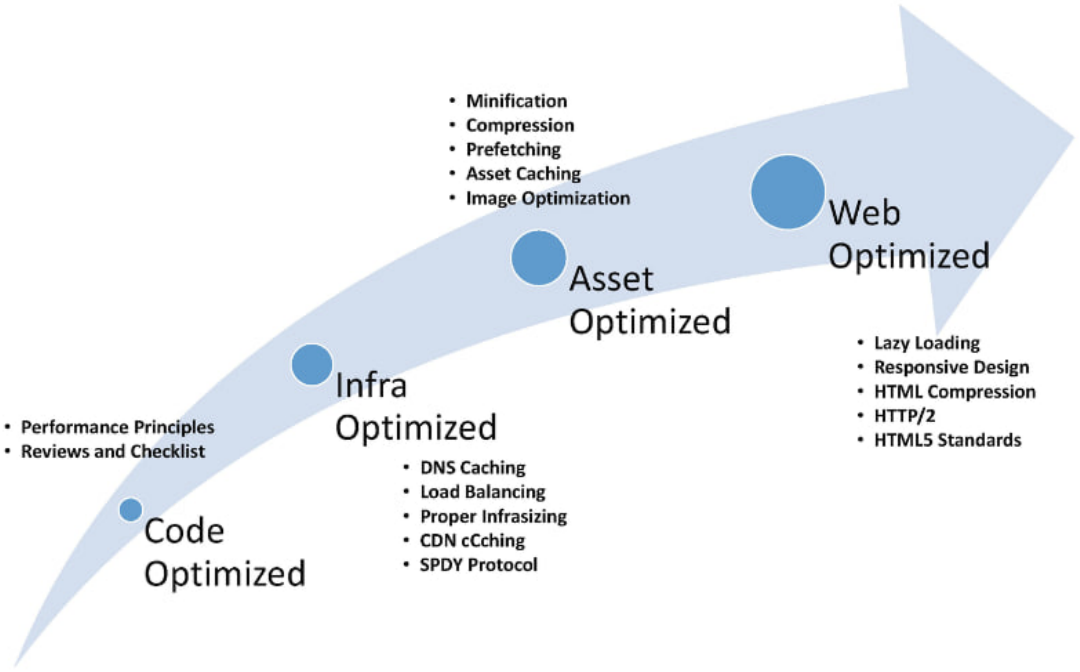


Рис. 2.1. Складові моделі оптимізації веб-додатку

Більш розширений перелік оптимізаційних параметрів включає в себе: веб-компоненти (HTML-контент, зображення/ресурси, сценарії, стильові таблиці), діяльність життєвого циклу проекту, обробка запитів, та управління продуктивністю. Кожен із цих параметрів має складові другого рівня, які також

впливають на продуктивність; наприклад, обробка запитів включає всі системи, що беруть участь в обробці веб-запиту на всіх рівнях [20].

Управління продуктивністю включає в себе метрики продуктивності, процеси продуктивності. Усі ці аспекти складають в купі загальну оптимізацію продуктивності веб-сайту.

Аналізуючи параметри оптимізації веб-додатку, Торстен Беєр сформував один із найбільш сучасних на сьогодні — комплексний та стратегічний підхід. Як підгрунтя наводяться безліч параметрів, вплив яких може бути дуже різним і досить непередбачуваним з першого погляду. Деякі з них піддаються прямому моніторингу, інші ж ледве можна грубо оцінити. Найбільші важелі оптимізації автор вбачає в сферах хостингу та обробки змісту веб-сторінки [21].

Метод Беєра вказує на те, що при створенні нового змісту сторінки або його редагуванні ресурсозатратними процесами є завантаження і відображення PDF-документів, зображень, шрифтів, відео- та аудіофайлів. Через це важливо ще на етапі створення контенту розглядати питання обсягу даних, оскільки “постоптимізація” завжди коштує дорожче ніж попередні розрахунки [21].

Основною проблемою для Беєра є те, що деякі веб-сайти надто сильно покладаються на використання систем управління контентом (CMS), що легко може призвести до завантаження зображень у невідповідних форматах, з неправильним масштабуванням або навіть по декілька разів. Це робить веб-сайти менш стійкими, повільнішими, а також непридатними до наступних оновлень.

Оптимізація та стиснення CSS-, JavaScript- і HTML-файлів мають трохи менший вплив, хоча все ще суттєвий [22].

Вирішення існуючої проблеми можливе на основі інструментів оптимізації, що, на думку науковця, криються в програмному коді і включають в себе [21]:

- Побудову чіткої та прозорої архітектури застосунку
- Оптимізацію взаємодії із DOM (DOM Manipulations)
- Застосування математично-ефективніших рішень мови програмування для взаємодії із даними
- Кешування та мемоізацію повторюваних даних

- Постійний моніторинг та обов’язкову пост-оцінку оптимізації

Частка ефективної оптимізації методами, що напряму не стосуються коду, як наприклад через мініфікацію збірки в сукупності не перевищує 10%, що свідчить про необхідність роботи над проектом “із середини”, а не ззовні [21, 23].

Що ж стосується хостингу, то це не тільки вибір правильного хостера, або ж місця розташування сервера для пришвидшення клієнт-серверної комунікації.

Дійсно слід мати на увазі, що ефективні методи оптимізації в першу чергу пов’язані із фізичною інфраструктурою. Наприклад, якщо сервер не підтримує стиснення Brotli або стандарт HTTP/2 чи HTTP/3, то це призводить до втрати можливості застосування ефективних інструментів оптимізації по принципу “out-of-box”, які можуть бути впроваджені всього за кілька хвилин просто шляхом автоматичної активації налаштувань хосту вручну. З іншого боку, деякі хости навіть не надають можливості визначити обсяг даних на веб-сторінці, не говорячи про більш нові стандарти [21, 24].

Розробка веб-застосунка з оптимізованою продуктивністю не можлива без дотримання рекомендацій та кращих практик на відповідному етапі життєвого циклу проекту.

Схожі ідеї лежать в основі методу WPO *(з англійської Web Performance Optimization - Фреймворк оптимізації продуктивності веб-сайтів)*, — підходу до розробки, орієнтованого на забезпечення відповідності конкретним, визначеним заздалегідь, вимогам. Це умовна конструкція, необхідна для вирішення різних аспектів оптимізації продуктивності веб-сайтів, вибору схеми досягнення цілей щодо продуктивності та її своєчасної оцінки [16].

Ідеєю застосування фреймворку передбачено організацію підтримки проекту таким чином, щоб він ґрунтувався на основі обраного архітектурного стилю, як правило стандартного Three-Tier Architecture, та містив вичерпний перелік етапів розвитку проекту та відповідних параметрів, призначених для оптимізації [25, 26, 27].

Однак для цілей даної роботи більш суттєвим є аналіз суто рекомендацій, що стосуються клієнтської частини проекту, завдяки чому ми скористаємось

типовими значеннями фреймворку, наведеними в таблиці та проаналізуємо їх більш детально.

Таблиця 2.1

Параметри оптимізації SPA фреймворку по категоріях

Архітектура та дизайн	Імплементація	Моніторинг
Розмір та потужність інфраструктури: ● апаратне забезпечення ● мережева інфраструктура ● системне програмне забезпечення Наприклад — наявні фізичні ресурси процесора або ж використання CDN чи характеристик веб-серверу	Вихідний код програмного забезпечення: ● загальні ● спеціальні Наприклад — компресія, кешування, масштабованість, управління процесом вичислення змін, збереження стану чи клієнт-серверної комунікації	Метрики Core Web Vitals: Largest Contentful Paint, First Input Delay, Cumulative Layout Shift, Page Loading Time, Time To First Byte, Speed Index, First Meaningful Paint
Змістове наповнення сторінки у веб-браузері: ● HTML-контент, зображення/ресурс, сценарії, стильові таблиці	Управління та збереження статичних файлів, мініфікація збірки: ● Медіа ресурси	Аналіз Google PageSpeed Insights та Chrome Dev Tools Panel
Моделювання стратегії продуктивності: ● Визначення параметрів продуктивності та механізму впливу на них	Адаптація “не-кодових” значень: ● Стратегії завантаження ● Безпека автентифікації Наприклад Lazy loading	Застосування інструменту Dom Size Analyzer

Продовження таблиці 2.1

Параметри оптимізації SPA фреймворку по категоріях

Архітектура та дизайн	Імплементація	Моніторинг
Визначення бажаних показників метрик продуктивності: ● Погодження допустимих метрик Core Web Vitals	Застосування обраних технік оптимізації продуктивності	Аналіз навантаження на фізичні ресурси у різних процедурних станах додатку

Відповідно до даних, наведених у таблиці Фреймворк розбиває оптимізацію на чотири підрівні, розподіляючи тягар поліпшення пропорційно та залежно від етапу існування проекту.

Аналогічно, засновники Фреймворку виокремлюють роботу над визначенням переліку допустимих показників перед початком, визначаючи які саме параметри є важливими, що можна покращити, а що взагалі не піддається сторонньому впливу. На стадії планування, важливо подбати, щоб майбутній проект мав достатньо фізичних ресурсів та рівномірно їх використовував, не перевантажуючи систему. Схожа тактика міститься в методі Беєра, де перелічуються аргументи за та проти інфраструктурних та апаратних рішень [21].

На особливу увагу заслуговує сценарій цілковитого поділу параметрів впливу на кодові та не кодові значення. Якщо “кодові” параметри уже згадувались в раніше розглянутих методах, то “не кодові” є унікальною особливістю підходу, суть якої полягає в застосуванні сторонніх сервісів, не пов'язаних із платформами розробки [26, 28].

Перевагою застосування Фреймворку є його стратегія тривалої підтримки. Після запуску проекту не менш важливим за аналіз та розробку функціоналу є моніторинг, адже навіть за умови дотримання рекомендацій можуть виникнути помилки та відхилення від погодженого плану відповідності. Підхід, закріплений

у Фреймворці не розділяє роботу проекту на до та після релізу, а наголошує на необхідності постійного поліпшення і втручання [20].

Підсумовуючи, у цьому розділі було розглянуто значення параметрів продуктивності веб-додатку, наведені та проаналізовані існуючі методи поліпшення продуктивності додатку в контексті часткового оновлення вмісту клієнтської частини веб-додатків. У подальшому ми перейдемо до математичної складової аналізу оптимізації.

2.2. Алгоритм покращення часткового оновлення вмісту веб-сторінки

Алгоритм покращення проблеми часткового оновлення вмісту веб-сторінки формується на основі проаналізованих параметрів, з урахування характеристик надійності, продуктивності та користувацького досвіду.

Для цього виразимо надійність, продуктивність та користувацький досвід крізь призму математичного моделювання, скориставшись відповідними математичними моделями, а саме моделлю Нельсона для категорії надійності, станами Маркова для категорії продуктивності та стандартом APDEX-у для користувацького досвіду.

Спочатку розглянемо модель надійності веб-додатку Нельсона.

Надійність роботи веб-додатку, а отже й користувацький досвід залежить від справності та безперебійності роботи застосунку. Надійність — це задумане, належне функціонування програмного забезпечення впродовж визначеного часу в певному середовищі [29].

Тоді, нехай P — програма, надійність якої слід проаналізувати, n — вхідний елемент програми, p(n) — ймовірність, що n обрано для P. Тоді $\alpha(n)$ — функція, яка вираховує можливість того, що програма виконується некоректно. Нехай, стан в якому програма повертає очікуваний результат визначатиметься як $\alpha(n) = 0$, тоді неочікуваний результат обраховуватиметься згідно з формулою, де функція $\alpha(n)$ матиме такий вигляд — $\alpha(n) = 1$.

Значить $\sum_{n \varepsilon D} \overline{p(n)\ \alpha(n)}$ (2.1) — ймовірність збою.

Врешті-решт, надійність програми визначається за формулою [29] —

$$R(P) = 1 - \sum_{n \varepsilon D} \overline{p(n)\ \alpha(n)} = \sum_{n \varepsilon D} \overline{p(n)(1 - \alpha(n))} \quad (2.2)$$

Оскільки тривалість часу перебування додатку на певному етапі свого життєвого циклу прямо вказує на те, наскільки ефективно виконуються процеси пов’язані із цим етапом, то можемо представити їх як сукупність станів в процесі Маркова. Крім того Марковська модель допоможе оцінити стійкість системи та виокремити слабкі місця програми.

Марков розглядав стани певного наукового явища як послідовності подій з випадковими переходами в будь-який момент часу. На відміну від гаусівських, пуассонівських процесів, процеси Маркова не досліджують поточні чи минулі стани та їх вплив на майбутній стан, адже не залежать ні від перших, ні від других [30].

Спершу визначимось із станами, у яких може перебувати досліджуваний SPA додаток. Для цього повернемось до рисунку 1.4, згаданого в першому розділі.

Життєвий цикл односторінкового додатку складається із таких стадій:

1. Завантаження веб-сторінки.
2. Навігації та переходу на наступну сторінку.
3. Відображення змісту.
4. Надсилання запиту та очікування відповіді.
5. Сигналізування про збій чи іншу помилку.

Позначимо кожен із цих станів n_i відповідно до порядкового номеру в переліку. Середній час перебування додатку в стані n_i складатиме $\frac{1}{\gamma_n}$ при чому γ_n — інтенсивність переходу з одного стану в інший, подана в матриці ймовірностей

переходу, яку можна встановити, зобразивши відношення станів у вигляді ланцюга Маркова, за прикладом, вказаним на рисунку нижче.

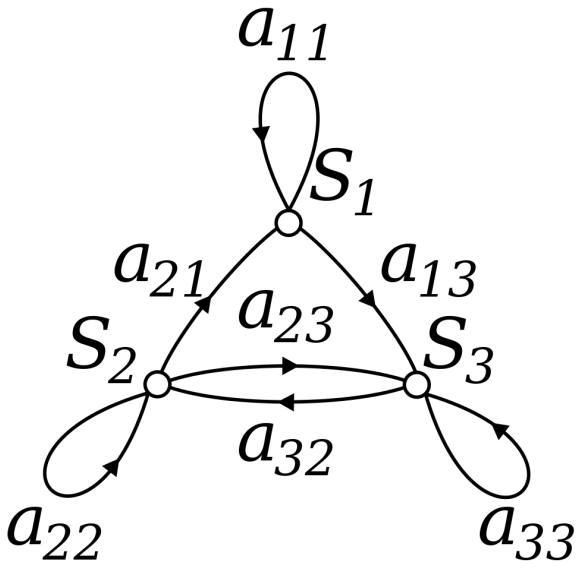


Рис. 2.2. Зразок ланцюга Маркова для 3-х станів

Пропускну спроможність системи та її завантаженість на основі параметрів станів Маркова можна визначити по формулах:

$$p_n = \gamma_n T_n, \quad (2.3)$$

де p_n - ймовірність перебування системи в стані n , T_n - середній час перебування в системі у стані n.

$$T_n = \frac{C_n}{T}, \quad (2.4)$$

де C_n ймовірність переходу зі n стану в інший стан, а T - середній час перебування в системі взагалі. [30]

Щоб визначити середній час очікування, завантаження, ймовірності надійності процесу необхідно врахувати часові параметри, отримані в ході

попередніх розрахунків та внести їх в матрицю переходів, проаналізувавши стани, в яких програма знаходиться найдовше, або дає збій. Таким чином ми аналізуватимемо критерій надійності.

Насамкінець розгляньмо стандарт APDEX-у (з англійської — Application Performance Index — індекс продуктивності додатку) — метрика, що використовується для оцінки, наскільки задоволений користувач, зважаючи на тривалість відгуку застосунку. Часто APDEX підміняється поняттям Quality of Service (*з англ. QoS- якість послуг*) [31]. В основу оцінювання покладено дослідження Нільсена щодо порогових значень часу відгуку застосунку на основі психології поведінки людини.

Згідно з дослідженням, для визначення рівня задоволеності користувача достатньо визначити емпіричні заміри часу відгуку додатку та обрати шкалу оцінювання категорій часу відгуку.

Оскільки різні додатки мають різні комерційні цілі та відповідно відмінності в навантаженні, то користувачеві надається можливість на основі особисто користувацького досвіду визначити рамки допустимості в межах мінімальних та максимальних значень, наприклад пройшовши опитування або залишивши загальний відгук.

В цьому випадку можна відрізняти такі межі [31]:

1. Верхня — Задовільно; Від 0 мсек. й до максимально прийнятного значення для користувача.
2. Середня — Толерантно; Від максимально прийнятного значення до часу, коли користувач починає задумуватись над тим, щоб покинути сторінку.
3. Нижня — Дратівливо; Абсолютно неприпустимий час очікування, після спливу якого, користувач закриває вкладку браузера.

Для кожного запиту веб-сторінки вираховуємо значення APDEX-у по формулі, позначивши кожен задовільний запит як r_s, кожен толерантний запит як r_t та знайшовши їх суму, розділену на кількість усіх запитів [31].

$$\text{APDEX} = \frac{\sum_{i}^{n}(r_s) + 0.5 + \sum_{i}^{n}(r_t)}{N} \quad (2.5)$$

Нільсен також пропонує власну шкалу оцінювання, виходячи із природи людської психології. Порівняємо отриманий результат з трьома пороговими значеннями нільсенської шкали [32]:

1. *До 0.1 секунди* або миттєва відповідь.

   Цей рівень відгуку створює відчуття безпосередньої маніпуляції процесом користувачем по принципу “тут і зараз”. Це ключова техніка для збільшення залученості трафіку додатку та гарантування відмінного відчуття взаємодії із сервісом.

2. *До 1 секунди або відносна відповідь.*

   Зберігається неперервність потоку думок користувача в процесі взаємодії. Користувачі можуть відчути певну затримку, але усвідомлення того, що комп'ютер генерує результат, дає відчуття контролю за процесом та певну свободу дій.

3. *До 10 секунд або повільна відповідь.*

   Увага користувача розсіюється. Користувачі починають зосереджуватись на інших речах, що ускладнює утримання їх уваги на першочерговому завданні, коли комп'ютер нарешті відповідає.

Узагальнюючи розглянуті в дослідженні математичні моделі, представляємо процес виконання алгоритму у вигляді розгорнутої покрокової схеми.

Алгоритм включатиме такі кроки:

1. Встановлення діапазонів допустимих значень метрик продуктивності веб-додатку, які б служили дороговказом для оцінювання ефективності оновлення вмісту;
2. Використання сторонніх сервісів, наприклад PageSpeed Insights (в основі яких знаходяться Core Web Vitals) або ж Chrome Dev Tools Performance для діагностики рівня користувацького досвіду.

3. Порівняння показників, отриманих в процесі діагностики із встановленими очікуваннями, звівши їх до спільного методу оцінювання;
    a. якщо очікування задоволені, то додаток оптимізований, а значить зміст сторінки уже ефективно оновлюється і наступним кроком буде завершення оптимізації;
    b. якщо показники далекі від діапазону очікування, то слід перейти до встановлення причин цього, в першу чергу розглядаючи додаток з точки зору поняття стабільності роботи, характеризуючи поведінку сторінки на основі станів Маркова.
4. Оцінка часових мірок перебування додатку в перехідних станах на основі математичної моделі Маркова:
    a. конкретизація усіх можливих станів, в яких може перебувати додаток, виходячи із природи його існування;
    b. вимірювання реальних часових проміжків станів додатку та фіксація найбільш часомістких станів;
    c. якщо сукупна та роздільна тривалість станів неприйнятна, тоді варто перейти до визначення наступного критерію,- надійності.
5. Аналіз надійності та передбачуваності роботи застосунку та ймовірність відмови, спираючись на модель Нельсона.
    a. Якщо додаток працює надійно, не повертає збоїв та має низький коефіцієнт помилок, проводимо повторну оцінку метрик.
    b. Якщо додаток повертає помилки, дає збій чи працює підозріло, то в такому випадку слід розрахувати коефіцієнт APDEX-у, переходячи до наступного кроку.
6. Знаходження середнього значення рівня стандарту APDEX-у
    a. Оцінити відгуки реальних користувачів та в разі незадовільного користувацького досвіду перейти до оптимізації.
    b. Підрахувати загальну сукупність зроблених запитів за певний проміжок часу та визначити, якою є частка толерантних та задовільних по відношенню до незадовільних.

c. Перейти до наступного кроку, розпочинаючи роботу над масштабною оптимізацією.

7. Розробка рекомендацій
   a. Оптимізація, виходячи із потреб покращень, встановлених під час оцінки рівня продуктивності, котра не обмежується загальними рекомендаціями.
   b. Застосування найкращих практик програмування, орієнтованих на інструменти розробки.

Результат розробленого алгоритму представлено у вигляді суцільної блок-схеми на рисунку 2.3.

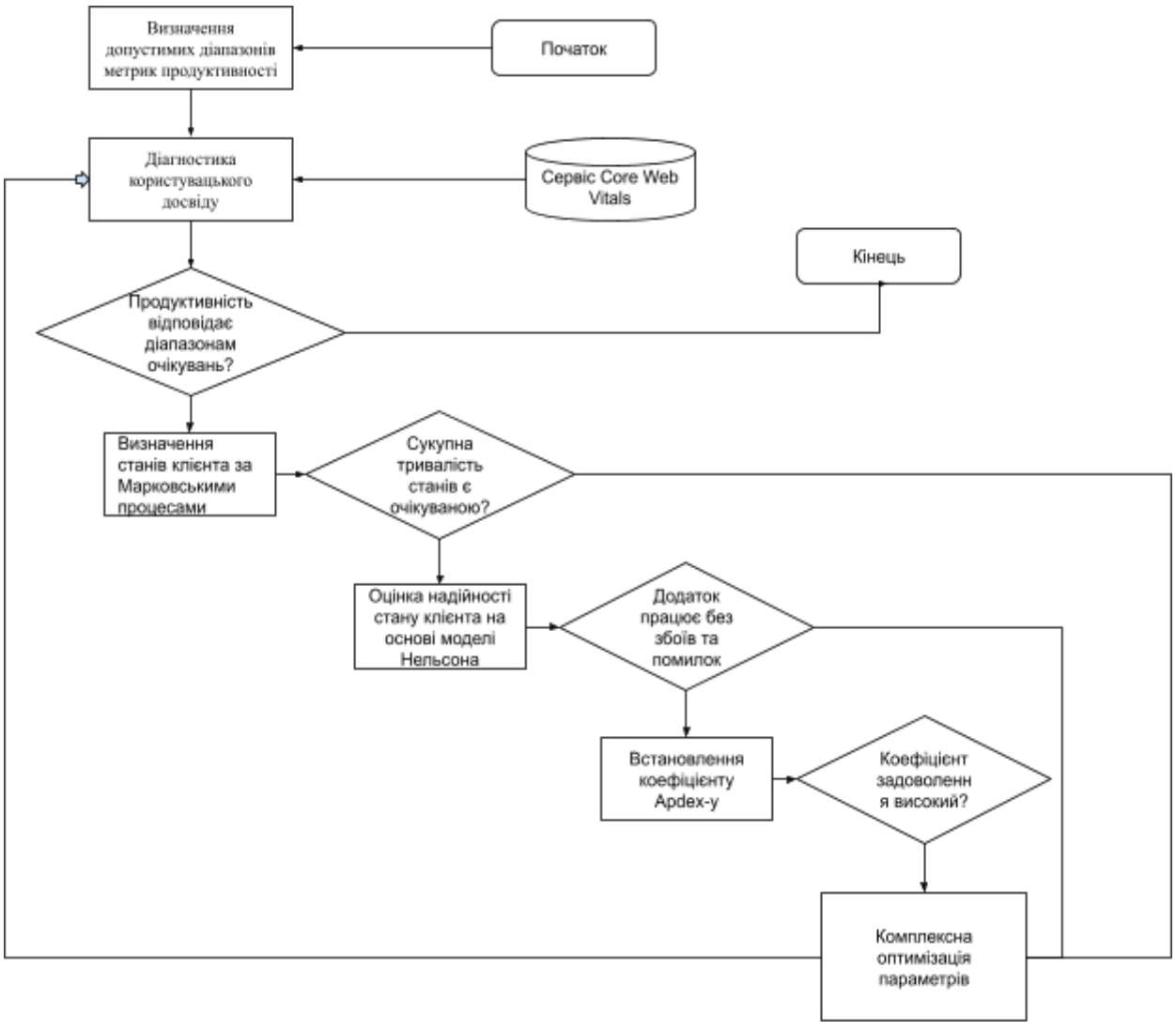


Рис. 2.3. Покрокова блок-схема алгоритму оптимізації

Резюмуючи, у цій частині роботи мова йшла про ідентифікацію параметрів, що впливають на дієздатність клієнтського веб-додатку та формують оболонку майбутнього механізму відображення електронного HTML-документу.

Окрім параметрів, досліджувалися методи впливу на продуктивність додатку на основі яких в кінці-кінців було математично вирахувано потребу оптимізації окремих процесів та створено алгоритм їх оптимізації як основу вирішенню поставлених в роботі завдань.

3 РОЗРОБКА РЕКОМЕНДАЦІЙ УДОСКОНАЛЕННЯ ПРОДУКТИВНОСТІ

З ціллю максимально звузити вплив сторонніх факторів на кінцеві показники ефективності застосування запропонованої в роботі оптимізації, було прийнято рішення про використання єдиної платформи розробки та тестування та розробки клієнтської частини веб-додатку із застосуванням найбільш популярних бібліотек в сучасному розробницькому середовищі.

Усі подальші кроки, зроблені в ході оптимізації, проведені на основі таких вихідних даних платформи розробки та тестування:

1. Веб-браузер: — 117.0.5938.88 (Official Build) (64-bit);
2. OS: Linux, Ubuntu 20.04 LTS (Focal Fossa);
3. CPU: Intel® Core™ i7-1165G7
4. RAM: 16 GB DDR4-2666 MHz RAM (2 x 8GB)
5. GPU: Intel® Iris® Xᵉ Graphics
6. Model: HP 15-dy2073dx

Що стосується програмного забезпечення, то для цілей роботи використовуються два ідентичні клієнтські додатки, розроблені на базі фреймворку Angular та бібліотеки React.js, розміщені в публічному репозиторії GitHub за посиланнями — https://2janko2.github.io/angular-deployment/, https://2janko2.github.io/react-deployment/ також наведені в додатках (лише основні компоненти програмного коду, з якими відбувається перетворення). Обидва додатки змодельовані на подобі соціальної мережі Conduit та є зразками практичних або інакше кажучи реальних додатків, які щоденно розробляються для комерційного застосування.

Angular та React.js два відповідні потужні інструменти для розробки веб-додатків, що мають різні підходи та можуть використовуватись у різних контекстах. Оскільки обидва інструменти послуговуються технікою часткового оновлення вмісту, то важливо показати яким власне є вплив інструмента розробки з його специфічними налаштуваннями [34, 35].

3.1. Визначення аспектів поліпшення функціоналу додатків на основі алгоритму

Перейдемо до оцінки потреби оптимізації на основі розробленого алгоритму. Діапазони допустимих значень показників впливу на оновлення сторінки наведені в таблиці 3.1.

Таблиця 3.1

Показники впливу на продуктивність веб-додатку та їх діапазони

№	Показники	Основні залежні параметри	Бажаний діапазон
1	Час завантаження сторінки	Визначення та підтягування лінивих модулів	До 300 мсек.
		Завантаження HTML-контенту	До 1 сек,
		Конструювання DOM-структури	До 500мсек
2	Відгук інтерфейсу користувача UI	Адаптивність інтерфейсу пристрою	до 1 мсек. (майже миттєво)
		Якість програмного коду	До 15 структурних одиниць згідно з моделлю Cyclomatic Complexity
		Комп’ютерна інфраструктура	$Max_x f(x)$

Продовження таблиці 3.1

Показники впливу на продуктивність веб-додатку та їх діапазони

№	Показники	Основні залежні параметри	Бажаний діапазон
3	Клієнт-серверна комунікація	Мережевий трафік	До 500мсек. $\frac{1}{K}$= С, але не більше 100
		Налаштування веб-серверу	Обсяг існуючих налаштувань
		Кешування та компресія	Обсяг збережених даних
4	Стабільність	Час доступу та час відновлення	99.5% від загального часу експлуатації програми
		Програмні помилки під час функціонування	$Min_x f(x)$
5	Використання ресурсів комп’ютера	Виконання програмних обчислювальних операцій	До 15% ресурсів системи
		Обробка графічних/медіа ресурсів	До 5% ресурсів системи
		Підтримка функціонування додатку в холодному стані	До 5% ресурсів системи

Ідентифікуємо середній час перебування додатків на всіх життєвих етапах та присвоємо цим етапам відповідні умовні позначення для візуалізації отриманих результатів:

Отже, стан, коли сторінка завантажується можна подати у вигляді позначення — S_1; стан, під час якого сторінка відображає готовий контент після завантаження — S_2; перебування в процесі навігації із однієї сторінки до іншої як S_3; надсилання запиту та очікування відповіді від сервера, як S_4; і нарешті стан збою роботи додатку,- як S_5.

Передбачуваною, вважається робота додатку, коли користувач переходить за наданим посиланням в веб-браузері та цим самим автоматично запускає додаток або переводить його в стан S_1, запускаючи завантаження сторінки у вікні браузера.

Після завантаження вікно браузера як правило відображає зміст головної сторінки на екрані та дає можливість користувачеві взаємодіяти із готовим функціоналом. Завдяки взаємодії користувача із додатком можливі раніше згадані переходи в інші стани.

Внаслідок взаємодії користувача додаток може перейти із стану відображення (S_2) в стан навігації (S_3) або в стан запитування додаткової інформації від веб-сервера (S_4).

В крайньому випадку якщо користувач вирішить оновити вкладку браузера, то додаток перейде зі стану відображення (S_2) в стан повторного завантаження сторінки (S_1). Із стану навігації (S_3) чи запитування (S_4) додаток врешті-решт повернеться в стан відображення (S_1).

Якщо відбудеться обривання стану запитування інформації (S_4), знову ж таки через оновлення, спричинене користувачем, можливий перехід в стан завантаження сторінки (S_1).

Беручи до уваги, що на кожному етапі життєвого циклу додаток може дати збій, а також після збою повернутись в будь-який із попередніх станів крім навігації, а також визначені раніше стани та переходи між ними, ланцюг Маркова матиме такий вигляд:

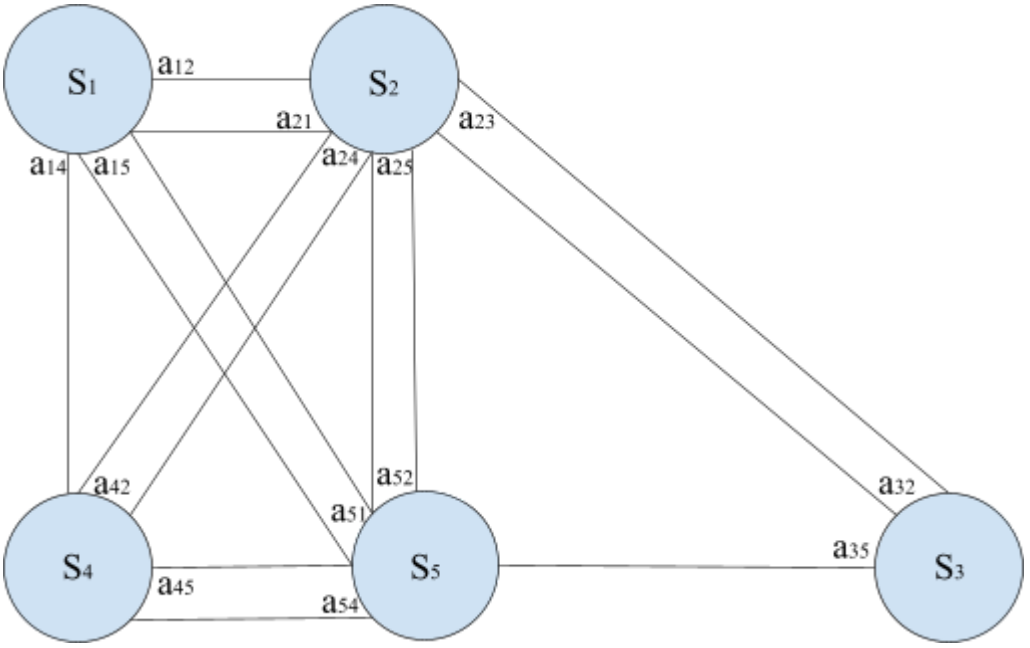


Рис. 3.1. Схема взаємозв’язків станів в ланцюгу Маркова

Виміряємо часові показники кожного із наведених в схемі станів, в яких обрані додатки можуть знаходитися час від часу, сконцентрувавшись лише на аналізі головної сторінки застосунку.

Оскільки доведено, що найбільше часу користувачі як правило проводять для ознайомлення зі змістом веб-сторінки, то для наочності аналізованих даних припустимо, що час перебування додатку в стані S_2 складатиме час прочитання. За даними Видавничого дому Києво-Могилянської Академії, доросла людина читає від 200 до 300 слів в хвилину, що становить 1000 — 1500 символів, враховуючи, що середньостатистичне слово складається із 5 літер [36].

Час перебування додатку в стані S_3 складатиме сумарний час переходу з головної сторінки по всіх посиланнях, розміщених на ній, на підсторінки поділений на кількість посилань.

Подібне стосується стану S_4, який складатиметься із сумарного часу обробки усіх запитів до веб-серверу з головної сторінки, розділеного на загальну кількість запитів.

Щодо стану S_5, то він дорівнює часу, коли роботу додатку повністю призупинено, або коли функціональність додатку для користувача дуже обмежена. Його розрахунок здійснюється як за рахунок тестування головної сторінки так і завдяки аналізу логів помилок.

Результати проведеного аудиту подані в таблицях нижче:

Таблиця 3.2

Показники марківських станів в React додатку

Стан	Середній час перебування React додатку в стані	Примітки
S_1	Від 2.3 до 3.7 мсек, в середньому 2.56 мсек.	4мс, 2.32мс, 1.38мс
S_2	Від 27 до 30 сек, в середньому 28.6 сек.	30с, 29с, 27с
S_3	Від 0.3 до 0.6 мсек, в середньому 0.41 мсек.	0.3мс,0.46мс, 0.47мс, 04мс
S_4	Від 0.9 до 1.6 мсек, в середньому 1.33 мсек.	1.52мс, 1.38мс, 1.25мс, 0.9мс
S_5	Значення відсутні	Значення відсутні

Таблиця 3.3

Показники марківських станів в Angular додатку

Стан	Середній час перебування Angular додатку в стані	Примітки
S_1	Від 2 до 2.8 мсек, в середньому 2.26 мсек.	2.8мс, 2мс, 2мс
S_2	Від 27 до 30 сек, в середньому 28.6 сек.	30с, 29с, 27с
S_3	Від 0.2 до 0.4 мсек, в середньому 0.3 мсек.	0.3мс, 0.22мс, 0.37мс
S_4	Від 1.4 до 2.5 мсек, в середньому 1.79 мсек.	1.58мс, 2.35мс, 1.45мс.
S_5	Значення відсутні	Значення відсутні

Після аналізу результатів, наведених в таблиці, підтверджується відмінність в роботі обраних для розробки платформ та потенційних місць “просідання” продуктивності. React додаток показав кращу роботу в стані S_4 , у той час як Angular,- в станах S_1 та S_3.

Слід зауважити, що час перебування додатків в усіх станах, крім S_2 відрізнявся. При цьому жоден з додатків не переходив в стан S_5, що є гарним свідченням надійності та стабільності роботи системи в цілому. Говорячи про стан S_5, то його значення суттєво перевищують всі інші показники разом взяті. Як уже згадувалось раніше, причина криється в самому сенсі роботи додатку, адже однією із його першочергових цілей є демонстрація сталої інформації, на ознайомлення з якою потрібен час.

Хоча проаналізовані показники в цілому не виходять за межі норми, вони все ще відстають від бажаних очікувань. І це лише те, що можна оцінити на основі марковських станів.

Наступним кроком відповідно до розробленого алгоритму є оцінка надійності додатку по формулі Нельсона. Оскільки надійність досить складний параметр, який не можна оцінити лише “сухо” по формулі, то скористаємось спершу open-source services (з англ. — сервіси, з відкритим вихідним кодом), створеними якраз для тестування стресостійкості та максимально допустимого навантаження.

Існує широка низка сервісів, які надають такі послуги або із застосуванням сторонніх скриптів, або ж прямо як окремий додаток із можливістю налаштування додатку під особисті потреби та вимоги користувача. Серед них — Apache JMeter, Gatling, Selenium Grid, Loader.io, BlazeMeter. Для тестування скористаємось останнім.

Перед запуском налаштуємо середовище тестування BlazeMeter. Для цього оберемо кількість користувачів, котрі одночасно нібито відвідуватимуть сайт впродовж певного часу, максимально інтенсивно досліджуючи його. Визначимо також CDN із наявного списку та вкажемо, що зв’язок повинен встановлюватись

саме через цю мережу. Також встановимо швидкість запитування сервера або ж оберемо максимальну кількість запитів.

З ціллю покращити точність отриманих результатів проведемо кілька ітерацій одна за одною для кожного із додатків.

Вхідні дані для проведених тестувань містяться в таблиці нижче.

Таблиця 3.4

Налаштування середовища тестування BlazeMeter

Параметри / Індекс	Кількість користувачів	Загальний час тестування	Швидкість запитування	CDN
1	10 чол.	10 хв.	10 з/хв	Europe-East
2	20 чол.	20 хв.	15 з/хв	Europe-East
3	30 чол.	30 хв.	30 з/хв	Europe-East
4	50 чол.	50 хв.	60 з/хв	Europe-East

За результатами тестувань не було виявлено випадків збою функціонування додатку незалежно від тягаря навантаження, що формується із параметрів налаштування наведених в таблиці. Крім цього не мали місце помилки у відображенні, завантаженні та обробці виду сторінки. Відсутні також системні помилки під час клієнт-серверної комунікації.

З іншого боку, завершені випробування вказали на визначені під час вимірювання показників продуктивності додатку за моделлю Маркова слабкі місця, підтверджуючи коректність оцінки на попередньому етапі. Серед них:

1. Зменшення середньої пропускної здатності;
2. Збільшення часу очікування відповіді;
3. Надмірний ріст споживання ресурсів локального комп’ютера;
4. Невнормована клієнт-серверна комунікація;

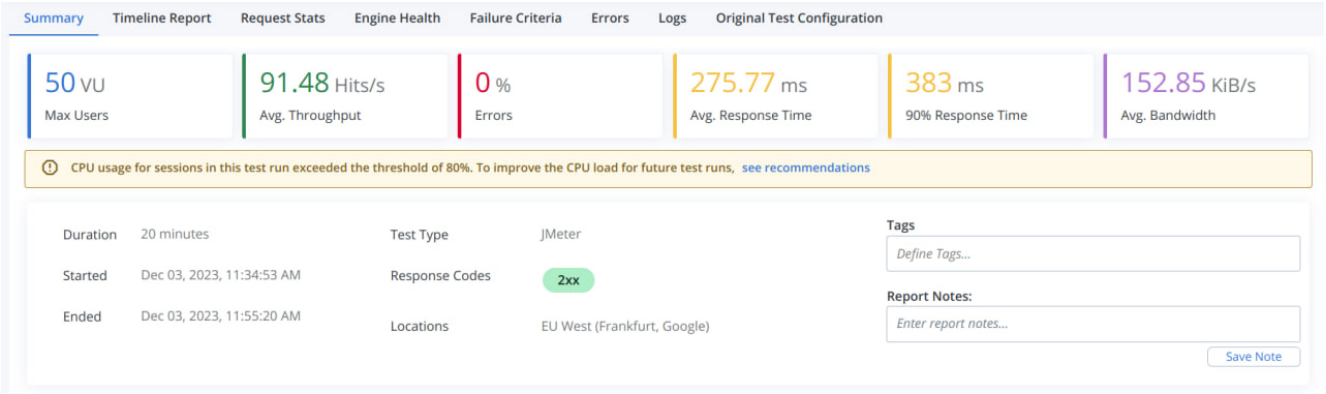


Рис. 3.2 Зразок панелі випробувань та даних порталу BlazeMeter

Таким чином, статистика відмов свідчить про їх повну відсутність, а отже повертаючись до формули надійності веб-додатку Нельсона, ми отримуємо надійну клієнтську частину.

Тепер перейдемо до оцінки заключної характеристики алгоритму. Через те, що категорія APDEX-у є найбільш суб'єктивною серед усіх та залежить від суджень відвідувача веб-сторінки та його/її досвіду, ми розмістили додатки публічно в GitHub Pages та запропонували вибірці респондентів пройти опитування та поділитись власним користувацьким досвідом, відповівши на декілька питань, після відвідування обох сайтів [37].

До вибірки було залучено близько 100 осіб з різним соціальним бекграундом, різних вікових та гендерних груп. Зважаючи на це, питання, що увійшли в редакцію опитування були максимально спрощені та висвітлені у зрозумілий для більшості осіб спосіб [37].

Опитування включає в себе такі питання:

1. Чи задоволені ви швидкістю відповіді сторінки та обробки ваших запитів додатком?
2. Наскільки швидкою на ваш розсуд була відповідь сторінки під час вашої взаємодії з нею?
3. Чи задумовувались ви покинути веб-сторінку внаслідок тривалості очікування на ваш запит?

4. Наскільки сильним було ваше бажання покинути сторінку під час очікування завантаження?

5. На який відсоток запитів, на вашу думку, час відповіді сторінки був допустимим?

На перше питання близько 70% усіх респондентів відповіли задовільно, тим самим показуючи, що в цілому користувацький досвід відповідає загальним вимогам, що ставляться до клієнтської частини додатку, однак той факт, що більше чверті опитаних все ще вбачають значний потенціал для покращення змушує однозначно вкотре вказує на оптимізаційний потенціал.

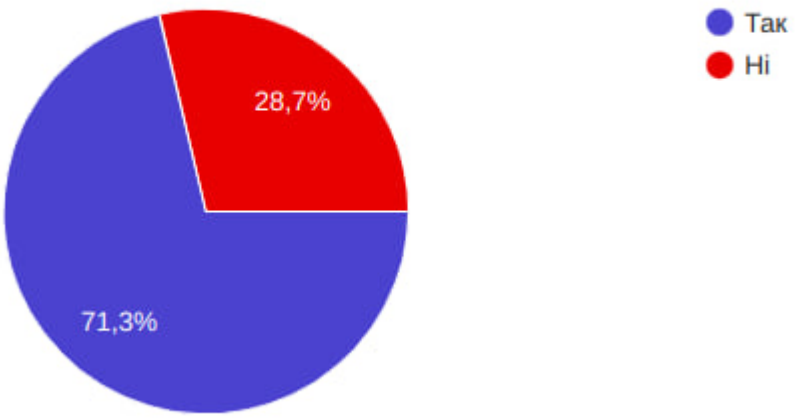


Рис. 3.3. Рівень задоволення швидкістю відповіді веб-сторінки

Питання з другого по четверте показали, що сторінка завантажується доволі швидко і лише 5% користувачів повністю були роздратовані якістю роботи функціоналу. 28.7% — визнають, що їм бракувало швидшого фідбеку, але в цілому також погоджуються з тим, що швидкість віддачі сайту допустима. 45.6% учасників заявили, що ловили себе на думці покинути принаймні один із аналізованих веб-додатків, при чому 43.6% з них визнають, що мали сильне бажання зробити це [37].

21.8% респондентів за підсумками останнього питання опитувальника вважають, що допустимими були менше ніж 75% усіх запитів, зроблених під час однієї сесії, тоді як 19.8% взагалі вказують на позначку менше 50% [37].

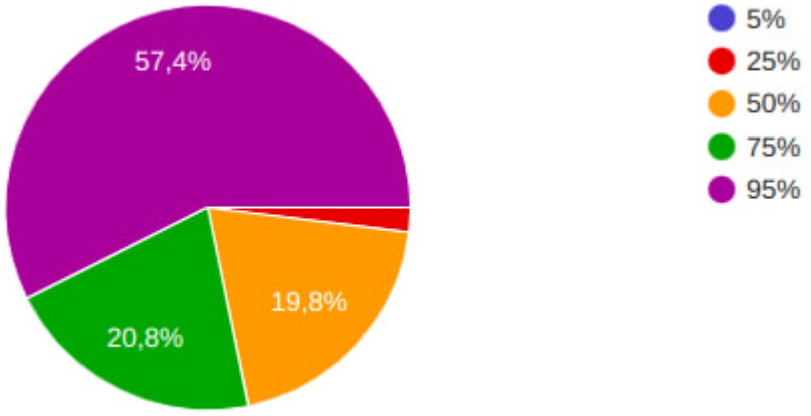


Рис 3.4. Рівень допустимості запитів

Для визначення згаданого індексу, розглядаємо за необхідне порівняти також дані отримані за рахунок власного тестування. З другого розділу пам’ятаємо, що чим швидше користувач отримує відповідь, тим вищим є коефіцієнт APDEX-у. Запити, впродовж 100 мсек вважаються толерантними. До 1 сек. — задовільними, а все, що більше 10 сек — неприпустимими [32].

Усього для кожного із додатків було здійснено 100 запитів, які розподілились таким чином [37]:

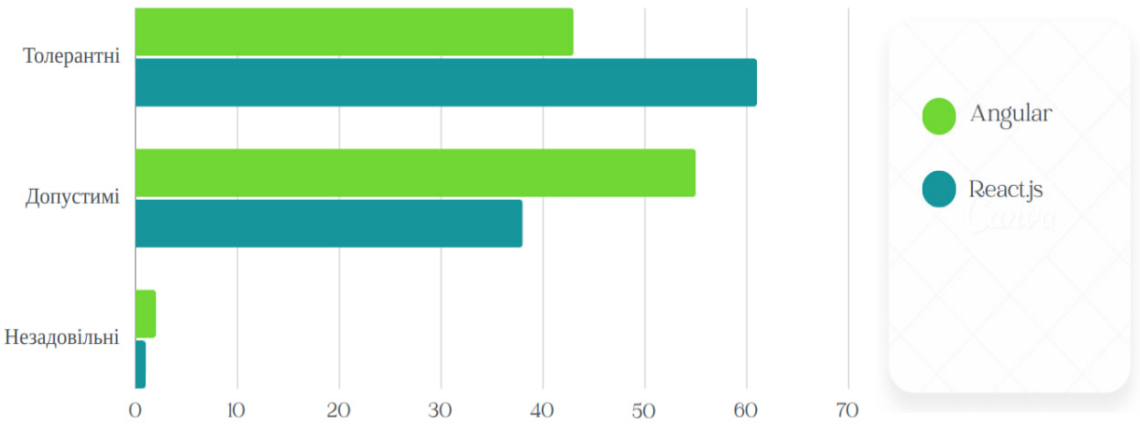


Рис 3.5. Розподіл запитів на основі користувацької оцінки

Відтак, відповідно до формули, стандарт APDEX-у складає 0.985 та 0.99 для Angular та React.js відповідно.

Підсумовуючи отримане в першій частині цього розділу, зазначимо, що для покращення функціонування часткового оновлення вмісту веб-сторінок аналізованих додатків слід оптимізувати роботу кожного із них у першу чергу, орієнтуючись на:

1. Скорочення загального часу переходу додатку із одного стану в інший.
2. Зменшення часу перебування додатків в окремих, найбільш часозатратних станах S_2, S_3, S_4.
3. Аналіз використання ресурсів сервера та його стресостійкістю під час збільшення навантаження на додаток.
4. Ідентифікація потенційних місць “просідання” клієнт-серверної комунікації
5. Встановлення причин незадовільного часу очікування фідбеку серверу та підвищення кількості запитів з толерантним часом відповіді

3.2. Проведення моделювання

Отримавши вичерпну інформацію про потенційні ризики для оновлення вмісту сторінок розглянутих додатків, ми нарешті можемо підійти до наступного

кроку, розробленого в алгоритмі оптимізації, а саме прямого впливу на джерело проблеми.

Для скорочення часу перебування додатку в конкретному стані ми пропонуємо розділений підхід, орієнтований на особливості стану, наведені попередньо.

Тривалість стану S_2 скоротимо за рахунок аналізу HTML-змісту на відповідність стандарту HTML5, компресії медіа-ресурсів, використання техніки лінивого завантаження, мініфікації коду та використання оптимованих шрифтів. Рівень впровадження стандарту HTML5 додатку доволі легко перевірити завдяки W3C Markup Validation Service, доступним за посиланням — https://validator.w3.org.

Отримавши звіт відповідності, замінимо застарілі HTML теги, відкоригуємо структуру елементів, котрі розміщені всупереч DOM конфігурації та наприклад не можуть належати до обраного піддерева.

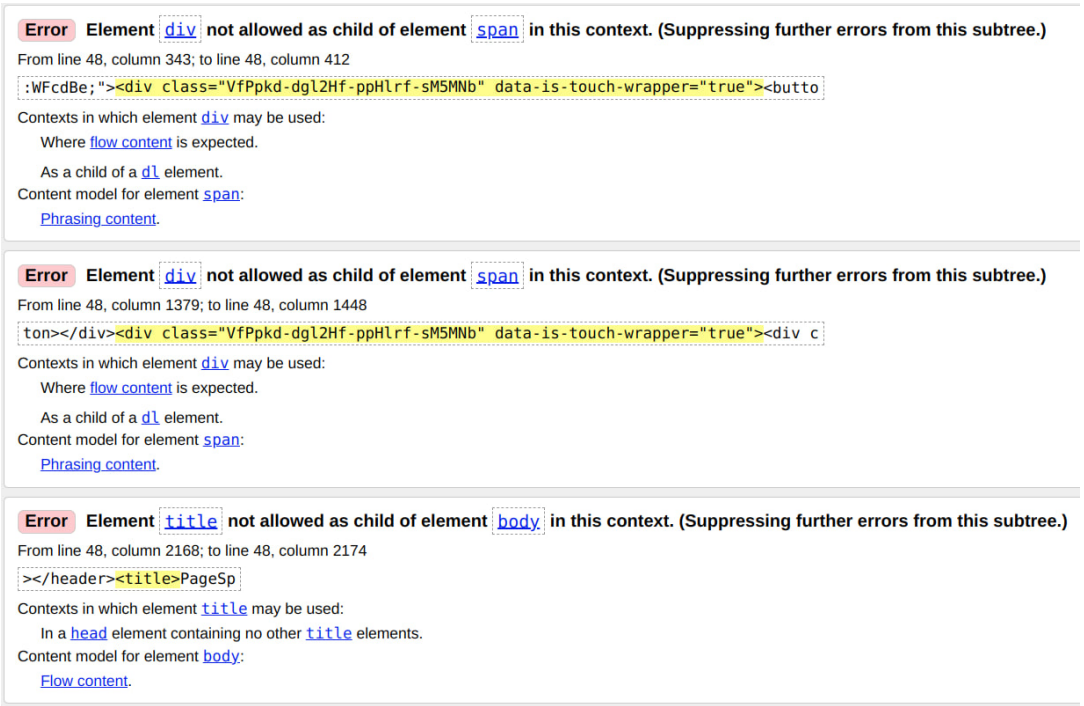


Рис 3.6. Зразок аналізу сервісом W3C Markup Validation Service

Обидва досліджувані додатки містять значну кількість подібних медіа-ресурсів, таких як картинки формату PNG, JPG, JPEG. В майбутньому цей

перелік може збільшитись в силу масштабування додатку і для того, щоб вручну не стискати кожну окрему картинку через онлайн-сервіси, скористаємось сторонньою бібліотекою, перед цим перевіривши, чи існують картинки в інших, оптимізованих форматах на зразок WebP чи Avif. Трансформуємо їх за допомогою сервісу compress-or-die.com.

Повертаючись до програмного вирішення, В React додатку для цієї цілі служитиме browser image compression, а в Angular вмонтований ngx image compress. Вони автоматично стабілізують вигляд картинки, зменшивши при цьому її вагу [38].

Схожим є принцип оптимізації використовуваних шрифтів. В додатках використовується шрифт "Source Sans Pro", sans-serif; підключимо в першу чергу його у форматі Woff2 та додамо локально до збірки. В HTML-розмітку внесемо тег <link> із атрибутом rel="preload" для попереднього завантаження.

На завершення, підключимо техніку лінивого завантаження, створюючи окремий модуль для лінивого завантаження та імпортуючи сторінки в центральному модулі Роутингу додатку з додаванням відповідної властивості path: 'lazy' для Angular та обертаючи кожен компонент за допомогою компонентів вищого порядку — HOC lazy для React.

На стан S_2 звичайно впливають також як таблиці стилів CSS, так і програмний код JS. Поліпшення програмного коду дуже містка задача, котра включає в себе як загальні підходи до оптимізації архітектури, вибору технологій, методів обробки даних, так і низку налаштувань, притаманних обраному інструменту розробки. Тому розглянемо тільки есенційні.

Умовно розділимо оптимізацію коду на спільне для обох додатків та відмінне. Тоді, спільне для обох платформ розробки включатиме в себе:

- Використання незмінюваних форматів даних [39]:
    1. Впровадження концепції незмінюваності даних та посилання на копії непримітивних типів даних стандарту ECMAScript 2015.

2. Використання існуючих методів об’єктів (додаток значною мірою складається саме з них) на зразок Object.freeze() для достовірності уникнення мутації даних.

- Усунення зайвих обчислень, які ніколи не використовуються [40]:
    1. Профілювання коду за допомогою сторонніх сервісів браузера, на зразок Chrome Dev Tools чи Firefox DevTools.
    2. Ін’єкція функціоналу Web Workers для раціонального розподілу потоків обчислень.

Відмінне для обох платформ розробки:

- Відмова від автоматичного порівняння змін DOM структури та налаштування алгоритму визначення змін:
    - *Angular-орієнтовані кроки:*
    1. Визначаємо компоненти, котрі отримують незмінні параметри або компоненти для чистого відображення та додаємо в їх налаштування ChangeDetectionStrategy.onPush
    2. З компонентів, які потребують постійного визначення змін, видаляємо ChangeDetectionStrategy.onPush, а в методах, які спричиняють зміну виду сторінки визначаємо this.changeDetectorRef.detectChanges()
    3. Якщо в компоненті відбувається запит на отримання інформації із сервера, то додатково користуватись функціоналом this.cdr.markForCheck()
    - *React-орієнтовані кроки:*
    1. Керування порівнянням змін методом shouldComponentUpdate для класових компонентів.
    2. Запам’ятовування вхідних даних компонентами вищого порядку React.memo та хуками UseCallback()
- Раціональний показ великих сукупностей даних:
    - Angular-орієнтовані кроки:
    1. Додавання до директив *ngIf та *ngFor унікальних функцій trackBy для чіткої ідентифікації HTML вузла в DOM дереві

2. Скорочення вживання *ngIf та заміна складних умовних конструкцій на ngSwitch

- React-орієнтовані кроки:

3. Генерування унікальних ID в компонентах, де візуалізуються дані та присвоєння їх властивостям key масивів елементів.

Результативність стану S_4 формується значною мірою на основі клієнт-серверної комунікації, через що ключ до його вдосконалення лежить у розробці стратегії якісної експлуатації API та клієнтської взаємодії з DAL (*з англійської - Data Access Layer - рівень доступу до даних*).

У такому разі спільними для обох платформ є подальші кроки:

- Налаштування кешування клієнтської частини веб-додатку
    1. Додати в API запити заголовки Cache-Control та Expires для визначення політики зберігання запитуваних даних на сервері та необхідності їх перевірки
- Взаємодія із браузерними сховищами
    1. Вибір підходящого сховища залежно від цілей зберігання інформації, localStorage або ж sessionStorage та додавання користувацьких даних, даних сесій, JWT токенів методом setItem();

Відмінними є кроки, описані нижче:

- Мінімізація передачі даних між клієнтом та сервером [41]:
    1. Заміщення REST API технологією GraphQl для отримання лише необхідних, запитуваних даних на стороні клієнту. Для цього в React-і слід передати в основний компонент додатку обгортку <ApolloProvider>, а в модульну оболонку Angular імпортувати модуль ApolloModule.
- Перехоплення та сортування запитів клієнтською стороною
    1. Імпортування готового функціоналу класу HttpInterceptor та налаштування його в @NgModule для Angular та встановлення бібліотеки axios та використання методу axios.interceptors у для React.

Показники надійності, встановлені на основі моделі Нельсона також піддаються поліпшенню завдяки корекції виконання програмного коду.

Знову ж таки спільними для React та Angular додатку будуть:

- Забезпечення ефективної обробки помилок
    1. Запровадження підходу Error Handling, де для кожного небажаного варіанту виконання коду передбачена стратегія реагування
    2. Обробка помилок нативними інструментами JS на зразок try{} catch (e){}, throwError, або ж new Error
- Регулярне оновлення та підтримка останніх версій збірки
    1. Налаштування періодичного автоматичного завантаження оновлень стеку технологій, що застосовується в розробці.
- Підключення Content Security Policy
    1. Достатньо розширити налаштування meta-тегу index.html додатку властивістю http-equiv із значенням Content-Security-Policy
- Перевірка та очищення даних
    1. Валідація введених користувачем даних у формах UI-компонентів шляхом використання валідаторів за замовчуванням, в тому числі Validators, PropTypes т.і.
    2. Верифікація параметрів запитів ще до відправлення запиту, а також обов’язкова серверна валідація.
- Візуальне тестування
    1. Вимагає інсталяції сторонньої бібліотеки, наприклад Storybook.js та створює окрему папку в корені проекту для тестування зовнішнього вигляду UI-компонентів.

Відмінне ж полягатиме в особливостях автоматизації методів тестування прогнозованості інструкцій програмного коду, зокрема для роботи із чистими функціями, технологіями збереження стану компоненту, а також перевірка вузіального співпадіння UI-компонентів із дизайном.

1. Unit-тестування - доступне out-of-box в обох додатках, проте для використання в Angular слід для кожного шаблону .ts створити

відповідний app.component.spec.ts та запустити команду ng test, в той час як для React.js потрібна конфігурація test.jsx та npm run test. (див. Додаток Х)

2. Збереження стану - використання персистентних хуків useState(), useReducer(), useSelector() для React.js та Services, BehaviorSubject, Variables для Angular.

Не слід забувати також про “не кодові” значення, згадані в попередньому розділі. Для надійності додатку їхню роль зазвичай недооцінюють або повністю ігнорують, вважаючи несуттєвою.

В досліджуваних додатках їх імплементація має обмежену роль, оскільки виключає ґрунтовну роботу із серверним потенціалом, наприклад резервне копіювання, стратегії аварійного відновлення, однак суто в розрізі front-end важливо пересвідчитись в перевірці прав доступу та в доступності додатку.

Таким чином у цьому розділі були розглянуті особливості оптимізації продуктивності веб-додатку на основі розробленого алгоритму поліпшення користувацького

4 ОЦІНКА ЕФЕКТИВНОСТІ ОПТИМІЗАЦІЇ

Оцінка ефективності розробленого алгоритму вирішення проблеми часткового оновлення вмісту — нелегке завдання, оскільки вимагає врахування значної кількості деталей та відповіді на питання, а яким власне інструментом слід користуватись, які метрики необхідно ранжувати, якою шкалою вимірювання зіставляти показники між собою та безліч інших питань.

З огляду на окреслені труднощі та обширність поставленого поняття було прийнято рішення звернутись до служб обслуговування веб-ресурсів, що є частиною знаменитих алгоритмів SEO оптимізації веб-браузерів. Підкреслюючи роль корпорації Google у розвитку ранжування оптимізаційних методів, найбільш популярними серед них є інструменти PageSpeed Insights та Chrome Dev Tools Performance [42, 43].

PageSpeed Insights — засіб перевірки швидкості та якості відображення змісту сторінки у виробничому режимі, що оцінює інтерактивність, завантаження та візуальну стабільність. В основі кінцевої оцінки знаходяться метрики [42, 44]:

1. Індексу швидкості (SI) — відображає швидкість від запуску завантаження до моменту, коли користувач побачить сторінку.
2. Час до першого байту (TTFB) — розраховує час між запитом та отриманням першого байту відповіді.
3. Час основного контенту (FMP) — вимірює час завантаження основного контенту, на який очікує користувач.
4. Блокування екрану (FID) — встановлює тривалість очікування користувача на можливість взаємодії із контентом.
5. Повне завантаження сторінки (PLT) — визначає сукупність загального часу повного завантаження сторінки.
6. Завантаження найбільшого елемента сторінки (LCP) — характеризує перебіг процесу появи найбільшого елементу контенту на сторінці.

7. Зміщення макету (CLS) — виявляє площу зсуву макету на основі порівняння дизайну

Chrome Dev Tools Performance — по своїй суті є вбудованим у браузер доповненням PageSpeed Insights, оскільки дозволяє записати події, що відбуваються під час вивантаження сторінки, аналізуючи більш глибокі параметри витрат ресурсів локального комп'ютера, наприклад — центрального процесора. Основні метрики Web Vitals та шкала їх оцінювання відображені на рисунку 4.1.

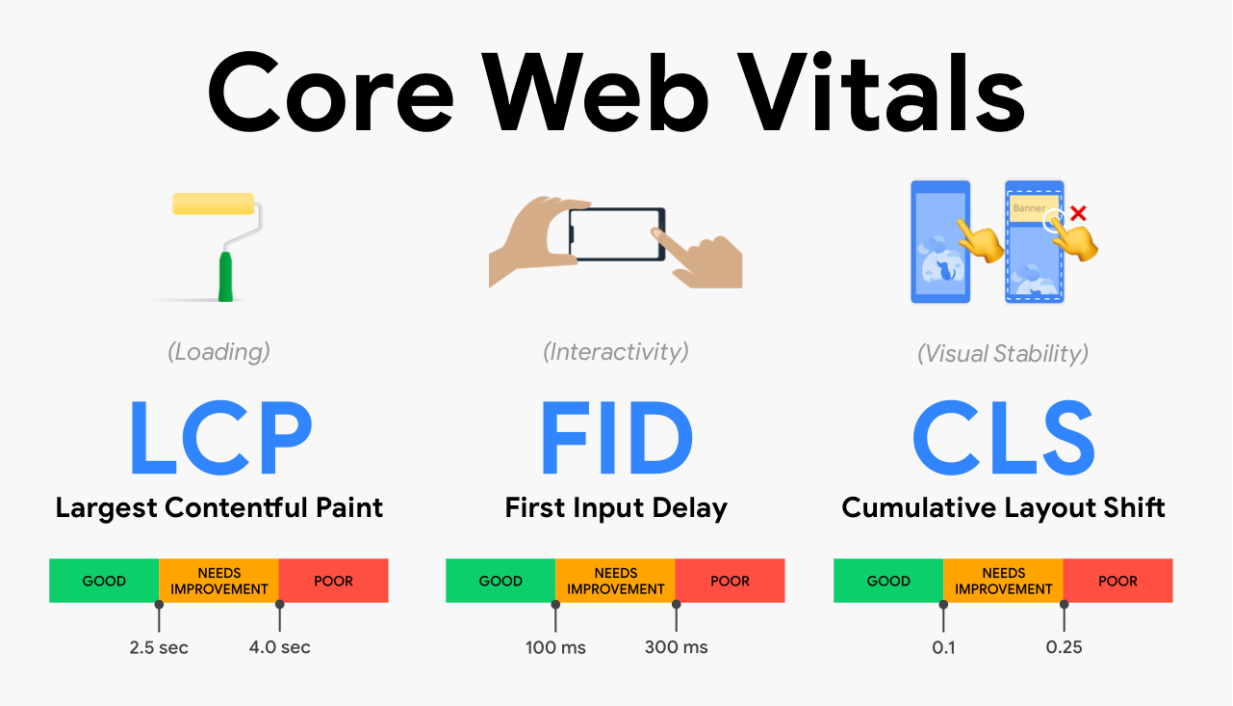


Рис. 4.1. Основні метрики Web Vitals

Досліджуючи рисунок 4.1. детальніше, бачимо, що для безперешкодного та дружелюбного до користувача (userfriendly) оновлення змісту сторінки вирішально, щоб завантаження відбувалось до 2.5 секунд, блокування інтерактивності складало не більше ніж 100 мілісекунд, а зсування не перевищувало 0.1 пікселя.

Наразі перейдемо до експертизи вихідних додатків. Дані, отримані в ході підрахунків для Angular та React.js десктоп додатку до впровадження розроблених рекомендацій подані на рисунках 4.2. та 4.3. нижче.

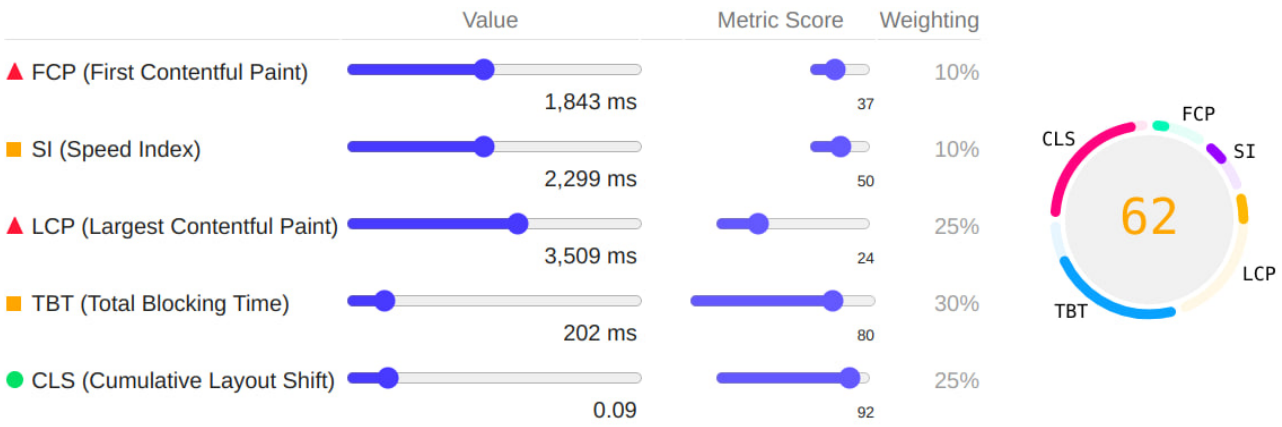


Рис. 4.2. Рівень продуктивності Angular додатку до оптимізації

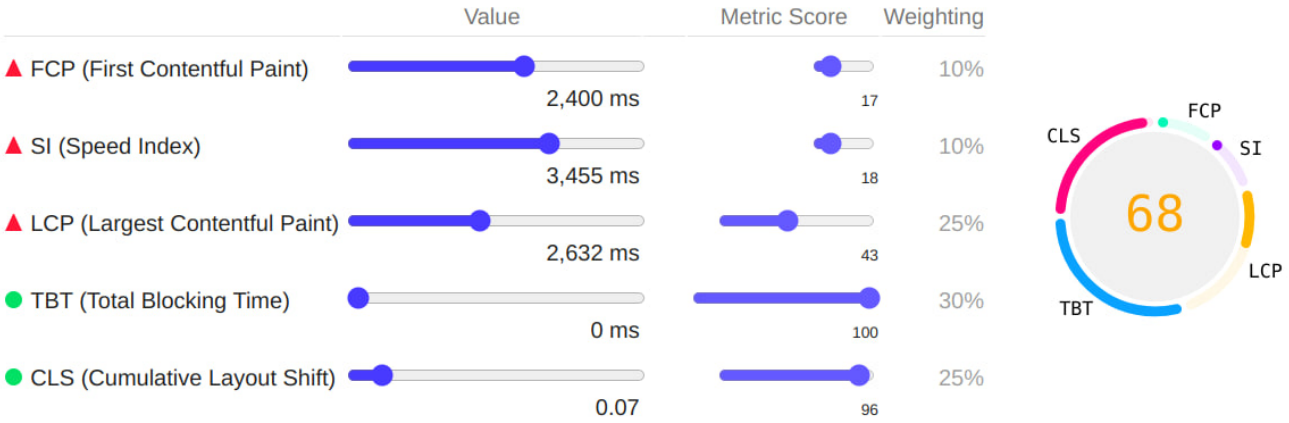


Рис. 4.3. Рівень продуктивності React.js додатку до оптимізації

Зіставляючи отримані на ілюстраціях відомості між собою, переконуємось, що метрики LCP, SI, FCP просідають в обох додатках, що в свою чергу проявляється в падінні загальної віддачі під час рендерингу сторінки з кожним новим оновленням. Сукупна вага трьох метрик складає 45% із яких більше половини залежить від значення параметру LCP. При цьому саме LCP є найгіршим з усіх відображуваних результатів шкали рівня оптимізації для Angular-десктопу та дорівнює 24/100.

Для React-десктопу це показник FCP із вагою в 17% із 100%. Загалом сервіс PageSpeed Insights оцінює продуктивність Angular додатку в 62%, а React.js додатку в 68% із 100% можливих, вказуючи на значний потенціал для покращення.

Справедливо зауважити, що використовуючи розроблений в роботі алгоритм оптимізації було встановлено схожі тенденції вразливостей та місць підвищеного ризику, що свідчить про спільний корінь проблеми, а також про дієвість обраного методу.

Впровадивши інструкції, висвітлені в алгоритмі та повторивши експертний огляд за допомогою сервісу ми пересвідчуємось в результативності використаного підходу, з урахуванням матеріалів рисунків 4.4. та 4.5.

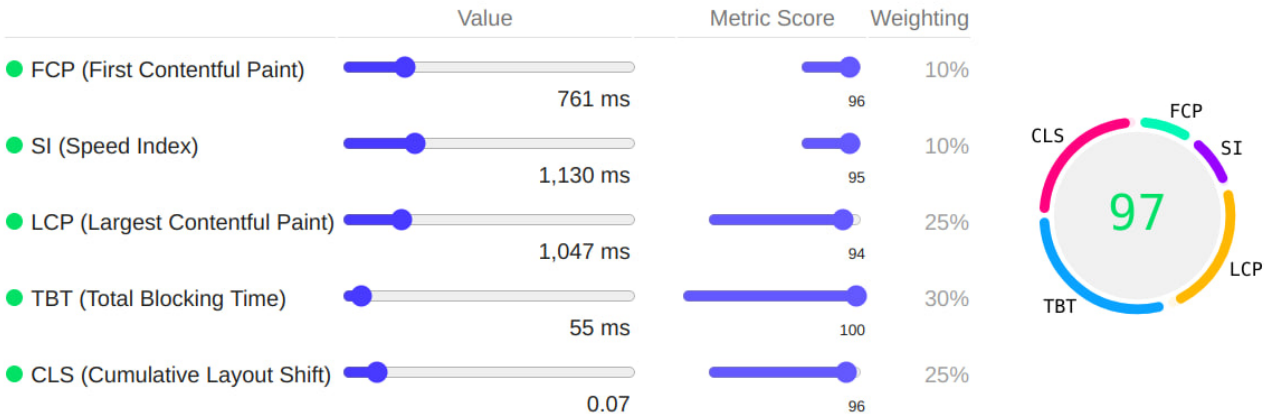


Рис. 4. 4. Рівень продуктивності Angular додатку після оптимізації

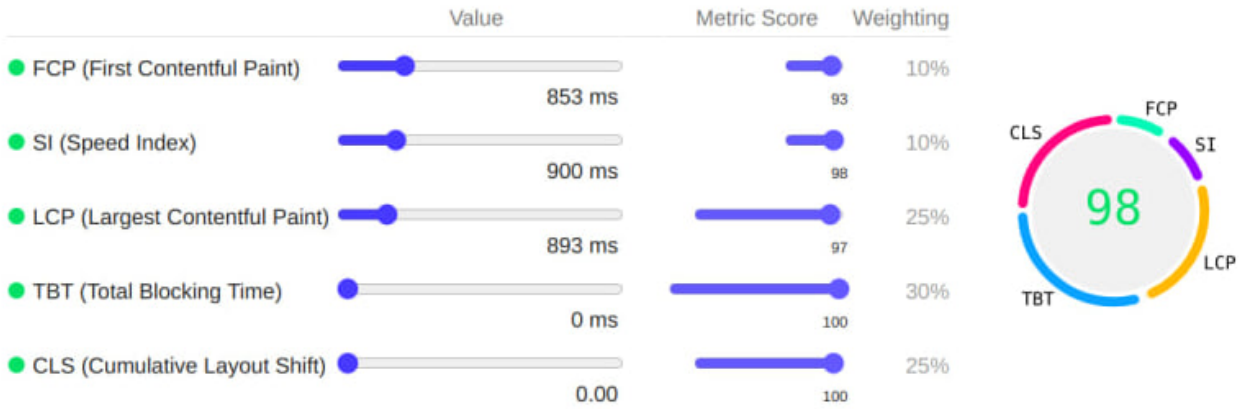


Рис. 4. 5. Рівень продуктивності React.js додатку після оптимізації

Переважна більшість метрик обох оптимізованих застосунків сягає більше 95%, а деякі навіть дорівнюють найвищому показнику ,— 100%. У відносному розрізі, ефект алгоритму складає близько 30-35%, хоча в абсолютному цей

показник значно вище, оскільки окремі його значення зросли більше ніж вдвічі, серед них:

1. LCP зріс з 24% та 43% до 94% та 100% відповідно.
2. FCP з 17% та 34% до 96% та 93%.
3. SI з 50% та 18% до 95% та 98%.

Експертиза PageSpeed Insights підтверджує той факт, що розробники часто випускають з виду важливість високої продуктивності веб-додатку, що як наслідок призводить до подальших проблем із механізмом оновлення вмісту веб-сторінки.

Крім цього сервіс PageSpeed Insights допоміг нам переконатись у важливості моніторингу параметрів продуктивності сторонніми сервісами, показуючи значний потенціал для визначення кореня проблеми та можливих підходів до його ліквідації.

Тепер скористаємось інструментом Dev Tools Performance, який надає розгорнутий звіт як по роботі окремих складових додатку, будь-то виконання окремих функцій, завантаження стилів, відображення контенту, так і загальний огляд функціонування збірки в цілому [46, 47].

Для цього, перебуваючи на сторінці веб-додатку, ми вмикаємо панель відладки (панель розробника або дебагу в залежності від обраного веб-браузера) та обираємо категорію Performance серед наведеного в меню переліку опцій. Натиснувши на кнопку запису ми короткостроково симулюємо очікувану діяльність користувача на сторінці, а по завершенню вимикаємо запис, розпочинаючи процес генерування отриманого звіту в консолі.

Враховуючи той факт, що звіт містить схожі дані, отримані в ході застосування сервісу PageSpeed Insights, сфокусуємося на особливостях завантаження, відображення, застосування JavaScript коду та на системних операціях, що виконуються впродовж нормального життєвого циклу застосунку, викладених на рисунках 4.6. та 4.7.

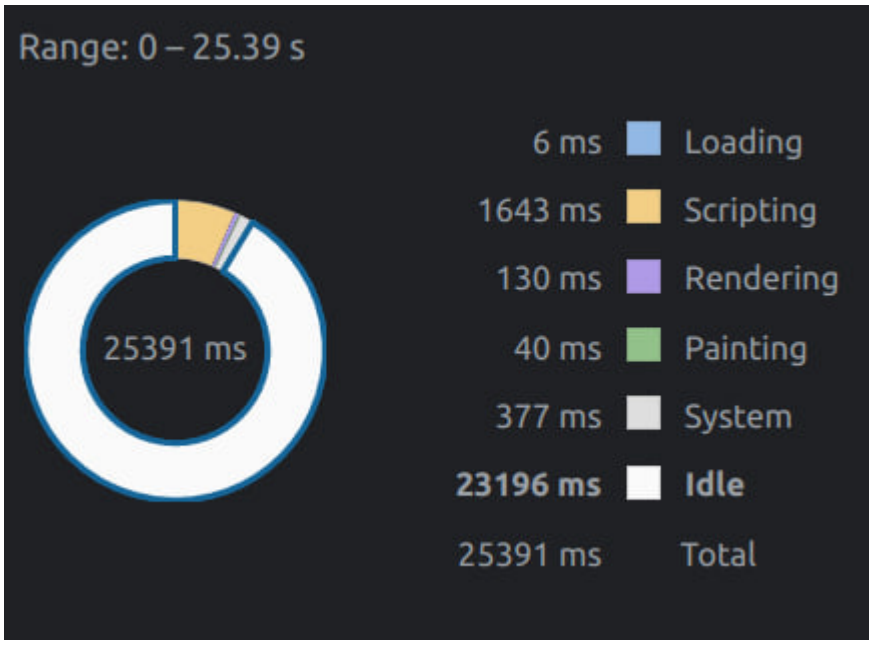


Рис. 4.6. Звіт Chrome Dev Tools Performance по оптимізованому React.js додатку

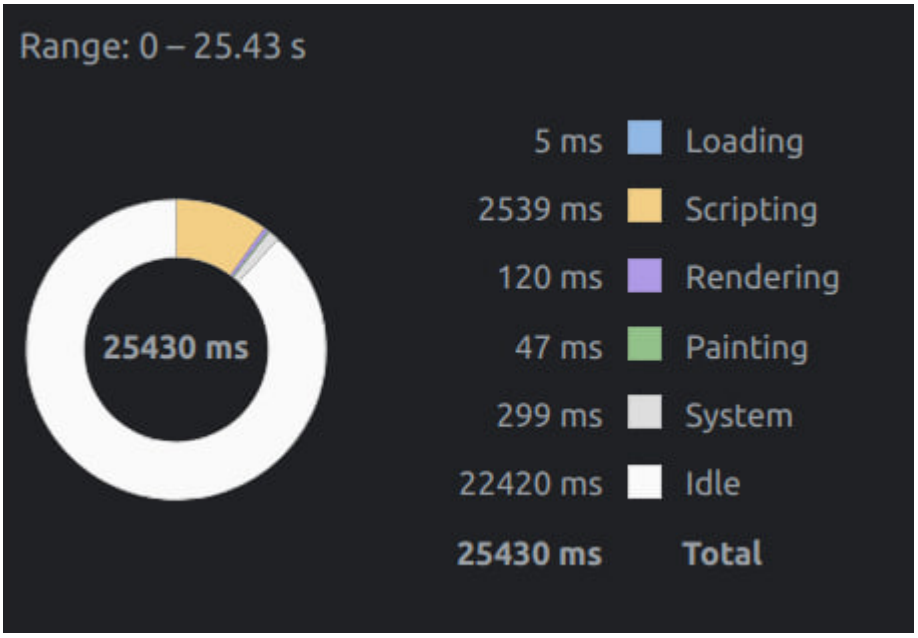


Рис. 4.7. Звіт Chrome Dev Tools Performance по оптимізованому Angular додатку

Упродовж 25 сек., відведених з урахуванням середнього часу, потрібного для ознайомлення із сторінкою, завантаження зайняло близько 5 мсек., створення відображуваного змісту до 120-130 мсек., процес малювання (фізичне зображення пікселів) 40-47 мсек., виконання системних операцій - 299-377 мсек., а виконання програмного коду 2.5 сек., решта - 22 сек. - простій, час відведений на прочитання змісту.

Отримані дані є прийнятними та не свідчать про недоліки та помилки в роботі додатків. При цьому важливо відмітити той факт, що одержані значення релевантні не лише для браузера Chrome; інші Dev Tools Performance Boards можуть варіюватись, проте похибка відхилення не має бути значною, якщо браузер є сумісним із застосованими веб-технологіями.

Узагальнюючи показники вимірюваних метрик, пересвідчуємось у дієвості та результативності створеного комплексного підходу до поліпшення роботи веб-додатку та стверджуємо про успішне досягнення поставлених в ході дослідження завдань та цілей.

ВИСНОВКИ

Механізм оновлення вмісту веб-сторінки має значний потенціал поліпшення рівня користувацького досвіду кінцевого продукту веб-розробки в сучасних умовах стрімкого технологічного прогресу. Його робота залежить як від складної взаємодії різних програмних, інфраструктурних, системних параметрів додатку, так і від особливостей обраних підходів в управлінні.

Якщо нехтувати такими факторами або залишати їх поза увагою, то ефект падіння продуктивності не змусить себе довго чекати. Як результат,- втрата зацікавленості користувачів, падіння репутації, доходу та непоправні наслідки. Саме тому, оптимізація оновлення вмісту веб-сторінки є актуальним та затребуваним дослідженням.

У дослідженні проведено комплексне порівняння технологій, що використовують техніку часткового оновлення вмісту веб-сторінки, визначено, що найбільш ефективним її застосуванням є використання підходу односторінкового застосунку. Встановлена залежність між продуктивністю роботи застосунку та якістю оновлення. Розглянуто методи впливу на параметри клієнтської частини веб-додатків, від яких напряму залежить підтримка високого рівня продуктивності.

Їх виявлення дозволило сформувати алгоритм оптимізації на основі математичних моделей категорій надійності, продуктивності, якості відображення, пріоритезуючи параметри відповідно до обсягу їх впливу на загальний рівень функціонування додатку. Визначено та сформовано кроки виконання алгоритму з розширеним поясненням послідовності застосованих рішень.

Практично реалізовано покращення часткового оновлення сторінки клієнтських веб-додатків на основі загальновідомих платформ розробки Angular та React.js в порівняльному ключі.

Насамкінець в роботі оцінюється ефективність застосування розробленого методу вирішення проблеми часткового оновлення вмісту сторінки завдяки використанню відповідних сторонніх сервісів, а також підтверджується

досягнення прогресу рівня продуктивності у відносному значенні 30% - 35% із можливих 100%, та в 2-3 рази в межах окремих абсолютних значень метрик Core Web Vitals.

Результати дослідження можуть бути використані для розробки програмного забезпечення як в комерційних цілях, так і з метою впровадження у освітньо-наукові та освітньо-професійні програми навчання вищих навчальних закладів, що спеціалізуються на викладанні технічних наук, зокрема мобільної та десктопної веб-розробки, веб-дизайну, або суміжних сфер.

ПЕРЕЛІК ПОСИЛАНЬ

1. Lohmeier L. (25.05.2023). Anzahl der Webseiten weltweit in den Jahren. Statista Search Department. [Infographic] [Електронний ресурс]. – Режим доступу: https://de.statista.com/statistik/daten/studie/290274/umfrage/anzahl-der-webseiten-weltweit/. Відвідано востаннє: 03. вер. 2023.
2. Бондарчук А.П., Хлиста І.А. Вирішення проблеми часткового оновлення вмісту веб-сторінки шляхом оптимізації параметрів SPA.V Всеукраїнська науково-практична конференція “Комп’ютерне моделювання та інформаційні технології” - Львів: НЛТУ, 2023. С. 286 - 291
3. Bondarchuk A. Khlysta I. Mitigating partial content update issues on web pages through SPA parameter optimization. State University of Information and Telecommunication Technologies. “Telecommunication and Information Technologies” 2023. P. 46 - 52.
4. Beglerović V., Pirija L., Prazina I. and Okanović V., "Detection and Logging Changes in Web Pages," 2022 21st International Symposium INFOTEH-JAHORINA (INFOTEH), East Sarajevo, Bosnia and Herzegovina, 2022, pp. 1-5, doi: 10.1109/INFOTEH53737.2022.9751305.
5. Irudayaraj, Prathap & P., Saravanan. (2019). Evolution of the Single Page Application in the modern web application development. Innovative Food Science & Emerging Technologies. 6. p. 141-145.
6. Fink, G.: (2014) Pro Single Page Application Development. Apress, New York
7. Scott E. A. Jr. (2015). SPA Design and Architecture: Understanding single-page web applications. Manning Publications Co. NY, USA.
8. Mukhiya, Suresh & Hung, Hoang. (2018). An Architectural Style for Single Page Scalable Modern Web Application. 5. 6-13.
9. Mesbah A., Van Deursen, A. (2007). Migrating multi-page web applications to single-page AJAX interfaces. In Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR

10. Gavrilă, Veronica & Bajenaru, Lidia & Dobre, Ciprian. (2019). Modern Single Page Application Architecture: A Case Study. Studies in Informatics and Control.
11. Solovei, V. & Olshevska, Olga & Bortsova, Y.. (2018). The difference between developing single page application and traditional web application based on mechatronics robot laboratory onaft application. Автоматизація технологічних і бізнес-процесів. 10. 10.15673/atbp.v10i1.874.
12. Edgar, M. (2023). Page Experience: Core Web Vitals and More. In: Tech SEO Guide. Apress, Berkeley, CA.
13. Bray T. (2003). Deep Linking in the World Wide Web. [Електронний ресурс]. – Режим доступу: https://www.w3.org/2001/tag/doc/deeplinking.html. Відвідано востаннє: 25 лис. 2023
14. Salomaa, J. (2020). Evaluating JavaScript Frameworks (Dissertation)[Online; accessed 27. Oct. 2023]. Available at: https://lnu.diva-portal.org/smash/get/diva2:1461878/FULLTEXT01.pdf
15. DataFlair (2023). JavaScript Libraries – You Must Know About [Електронний ресурс]. – Режим доступу: https://data-flair.training/blogs/javascript-libraries/. Відвідано востаннє: 3 гру. 2023
16. Puskaric, Hrvoje & Djordjevic, Aleksandar & Stefanovic, Miladin & Zahar Djordjevic, Marija. (2019). Development of web based applications using spa architecture. Proceedings on Engineering Sciences. 1. 457-464. 10.24874/PES01.02.044.
17. Selakovic M. and Prade M., (2016). Performance Issues and Optimizations in JavaScript: An Empirical Study. IEEE/ACM 38th International Conference on Software Engineering (ICSE), 2016, pp. 61-72, doi: 10.1145/2884781.2884829
18. Krishnamurthy Balachander, Wills Craig E. (2000). Analyzing factors that influence end-to-end Web performance, Computer Networks, Volume 33, Issues 1–6, 2000, Pages 17-32, ISSN 1389-1286, https://doi.org/10.1016/S1389-1286(00)00067-0
19. Wagner, J. (2016) Web Performance in Action: Building Fast Web Pages Manning Publications. NY, C. P.-316

20. Shivakumar, S.K. (2020). Getting Started with Web Performance Optimization. In: Modern Web Performance Optimization. Apress, Berkeley, CA.
21. Beyer T. (2023). Praktischer Leitfaden zur Prüfung und Optimierung – mit zahlreichen Tool-Tipps und Programm-Codes. Springer Gabler Wiesbaden. https://doi.org/10.1007/978-3-658-41093-3
22. Van Riet J., Malavolta I. (2019). "Client-side Performance of Web-based Applications: the State of the Art." [Online; accessed 12. Oct. 2023]. Available at: https://jaspervanriet.nl/assets/literature_study.pdf.
23. Hussain, Azham & Phd, Mohammad & Mohamad Tahir, Hatim. (2015). Industrial Web Application Customization Mechanism to Improve Software Quality and Productivity. International Journal of Mathematical Models and Methods in Applied Sciences. 9. 396-403.
24. Friedrich. C., Leucker M. (2016). Load testing of web applications and corresponding optimizations. [Online; accessed 23. Sept. 2023].Available at: https://www.isp.uni-luebeck.de/sites/default/files/main.pdf
25. Слабінога, М. О., & Чабан, С. В. (2022). Розробка веб-додатків в контексті оптимізації їх швидкості. Таврійський науковий вісник. Серія: Технічні науки, (3), ст. 63-69
26. Клушин Ю. С., Захарчин Ю. Б. (2020). Підвищення швидкості роботи веб-додатків. Випуск 2, номер 1, Науковий журнал "Комп’ютерні системи та мережі".
27. Курко, В. С., Колесник, І. С., Боровська, Т. М. (2021). Метод підвищення швидкості завантаження веб-сторінок. Оптико-електронні інформаційно-енергетичні технології, 41(1), 13–19. https://doi.org/10.31649/1681-7893-2021-41-1-13-19
28. Ярош, О. Л., & Бабаков, Р. М. (2023). Методи оптимізації продуктивності веб-застосунків. Прикладні інформаційні технології, 87-89.
29. Thayer T.A., Lipow M., Nelson E.C., Software Reliability. New York: North-Holland, 1978. https://doi/10.1145/1010832.1010859

30. Giambene G. (2005). Markov Chains and Queuing Theory. In: Queuing Theory and Telecommunications. Springer. - https://doi.org/10.1007/0-387-24066-7_5
31. Apdex Users Group [Електронний ресурс]. – Режим доступу : https://www.apdex.org/. Відвідано востаннє 10. гру. 2023
32. Nielsen J., (1993). Response Times: The 3 Important Limits. [Електронний ресурс]. Режим доступу : https://www.nngroup.com/articles/response-times-3-important-limits/. Відвідано востаннє 1. гру. 2023.
33. Lipiński, Adam & Pańczyk, Beata. (2023). Performance optimization of web applications using Qwik. Journal of Computer Sciences Institute. 28. 197-203. 10.35784/jcsi.3672.
34. Kaluža, M., Troskot, K. i Vukelić, B. (2018). Comparison of front-end frameworks for web applications development. Zbornik Veleučilišta u Rijeci.
35. Щербаков Є.В. Дослідження ефективності використання бібліотеки React / Є.В. Щербаков, М.Є. Щербакова [Електронний ресурс]. – Режим доступу : http://nvdu.snu.edu.ua/wp-content/uploads/2022/08/2022-23-7.pdf. Відвідано востаннє 14. жов. 2023.
36. Судакова О. (2016). Видавничий Дім НаУКМА. Швидкість читання [Електронний ресурс]. – Режим доступу : http://publish-ukma.kiev.ua/ua/novini/71-10-pidkazok-dlya-shvidkogo.html Відвідано востаннє 03. гру. 2023.
37. Хлиста І. (2023). Опитування задоволеності користувачів досвідом взаємодії з додатками React.js та Angular до впровадження оптимізації [Електронний ресурс]. – Режим доступу : https://docs.google.com/forms/d/e/1FAIpQLSdVdiFr5znm_zW9ADd2ANHWT48ExZmvYUBeLVz16N6iGo3rrw/viewform. Відвідано востаннє 12. гру. 2023.
38. Javeed, A. (2019). "Performance Optimization Techniques for ReactJS," IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT), Coimbatore, India, 2019, pp. 1-5, doi: 10.1109/ICECCT.2019.8869134.

39. Hakulinen R. (2018). Using immutable data structures to optimize angular change detection. [Online; accessed 25. Aug. 2023]. Available at: https://trepo.tuni.fi/bitstream/handle/123456789/26217/Hakulinen.pdf
40. Golestani H., Mahlke S. and Narayanasamy S., "Characterization of Unnecessary Computations in Web Applications," 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Madison, WI, USA, 2019, pp. 11-21, doi: 10.1109/ISPASS.2019.00010.
41. Vesić, Milena & Kojić, Nenad. (2020). Comparative analysis of web application performance in case of using REST versus GRAPHQL. 17-24. 10.31410/ITEMA.2020.17.
42. Google Pagespeed Insights Lighthouse 10, (2023). How the Performance score is weighted, [Online; accessed 5. Aug. 2023]. Available at https://developer.chrome.com/docs/lighthouse/performance/performance-scoring/#lighthouse-10
43. Vesper (2018). Measuring time-to-interactivity for modern web pages. In: Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation, NSDI. USENIX Association. Available at: https://www.cs.princeton.edu/~ravian/publications/vesper_nsdi18.pdf
44. Hakim M. (2016). Speed index and critical path rendering performance for isomorphic single page applications [Online; accessed 3. Sept. 2023]. Available at https://cs.winona.edu/cs-website/current_students/Projects/CSConference/2016conference.pdf
45. Anson E. (2021). Was bedeutet das Fehlen von Felddaten für Ihre Core Web Vitals? [Електронний ресурс]. – Режим доступу :https://www.ezoic.com/de-lang/was-bedeutet-das-fehlen-von-felddaten-fur-ihre-core-web-vitals. Відвідано востаннє: 02. лис. 2023
46. Skvorc, B. et al. (2018) The Web Performance Collection. 1st edn. SitePoint.
47. Vasilijević, Vasilije & Kojić, Nenad & Vugdelija, Natalija. (2020). New approach in quantifying user experience in web-oriented applications. 9-16. 10.31410/ITEMS.2020.9.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

«ВИРІШЕННЯ ПРОБЛЕМИ ЧАСТКОВОГО ОНОВЛЕННЯ ВМІСТУ ВЕБ-СТОРІНКИ ШЛЯХОМ ОПТИМІЗАЦІЇ ПАРАМЕТРІВ SPA»

Виконав: студент групи ПДМ-62 Хлиста Іван Андрійович

Керівник: д.т.н., професор кафедри ІПЗ Бондарчук Андрій Петрович

Київ - 2024

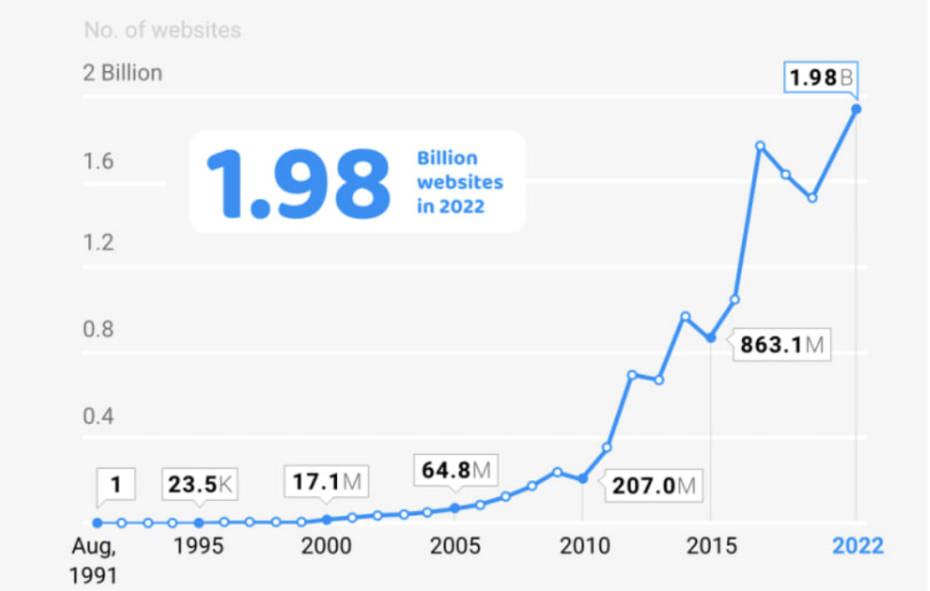
МЕТА, ОБ’ЄКТА ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: покращення користувацького досвіду шляхом вирішення проблеми часткового оновлення вмісту веб-сторінки за рахунок оптимізації її параметрів.

Об’єкт дослідження: процес часткового оновлення вмісту веб-сторінки клієнтської частини веб-додатку

Предмет дослідження: методи, що впливають на часткове оновлення клієнтської частини сторінки

АКТУАЛЬНІСТЬ РОБОТИ



Джерело: internetlivestats

3

ПРИКЛАДИ ЗАСТОСУВАННЯ ТЕХНІКИ ОНОВЛЕННЯ ЗМІСТУ

Х-ка	Традиційні	MPA	RIA	SPA
Особливість	Багато сторінок, що по черзі перезавантажуються	Багато сторінок, що разом перезавантажуються	Або одна сторінка, або безліч	Одна сторінка, що оновлює безліч елементів
Завантаження (ініціалізація)	Кожна сторінка наново	Кожна сторінка наново	Одноразово	Один раз при запуску, після чого часткові оновлення.
Оновлення	Цілковите перезавантаження сторінки на запит	Цілковите перезавантаження сторінок на запит	Оновлення однієї сторінки внаслідок AJAX запиту	Ajax-запити для підвантаження даних без перезавантаження сторінки.
Розробка	Серверні мови програмування	Фреймворки та бібліотеки	UI Бібліотеки	Фреймворки та бібліотеки
Споживання ресурсів	Ресурсоємкий та вимогливий	Дуже вибагливий, вимагає багато рес.	Менше ніж MPA, більше ніж SPA	Мінімум
Швидкість оновлення виду сторінки	Низька	Низька	Висока	Висока
Рівень освоєння	Високий	Високий	Середній	Середній
Автоматизація	Часткова	Часткова	Часткова	Часткова

4

ОСНОВНІ ПАРАМЕТРИ ВПЛИВУ НА ПРОДУКТИВНІСТЬ

№	Категорія	Метрика	Основні залежні параметри
1	Час завантаження сторінки	FCP, TBT, LCP	Завантаження HTML-контенту
			Конструювання DOM-структури
			Визначення та підтягування необхідних модулів (Lazy Loading)
2	Відгук інтерфейсу користувача UI	SI, TBT, CLS	Адаптивність інтерфейсу пристрою
			Якість програмного коду
			Комп'ютерна інфраструктура
3	Клієнт-серверна комунікація	TBT, FCP, SI	Мережевий трафік
			Середній час відповіді запитів
			Кешування та компресія
4	Використання ресурсів комп'ютера	LCP, TBT, SI	Виконання програмних обчислювальних операцій
			Обробка графічних/медіа ресурсів
			Підтримка функціонування додатку в холодному стані

5

МАТЕМАТИЧНА МОДЕЛЬ ОЦІНКИ ПАРАМЕТРІВ SPA

1. Оцінка параметрів надійності, стабільності, стресостійкості

$$R(P) = 1 - \sum_{n \varepsilon D} \overline{p(n)\,\alpha(n)} = \sum_{n \varepsilon D} \overline{p(n)(1 - \alpha(n))} \quad ,$$

де P — додаток, який слід проаналізувати,

n — вхідний елемент програми,

p(n) — ймовірність, що n обрано для P.

a(n) — функція, яка вираховує можливість того, що програма виконується некоректно.

Якщо, a(n) = 1, то отримується неочікуваний результат.

6

МАТЕМАТИЧНА МОДЕЛЬ ОЦІНКИ ПАРАМЕТРІВ SPA

2. Оцінка параметрів, що формують користувацький досвід

$$\text{APDEX} = \frac{\sum_{i}^{n}(r_s) + 0.5 + \sum_{i}^{n}(r_t)}{N} \quad ,$$

де N - загальна кількість запитів,

0.5 - коефіцієнт часткового задовільних,

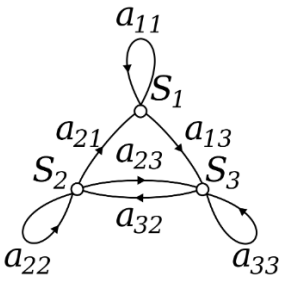
$\sum(r_s)$ - кількість задовільних відповідей,

$\sum(r_t)$ - кількість толерантних відповідей.

7

МАТЕМАТИЧНА МОДЕЛЬ ОЦІНКИ ПАРАМЕТРІВ SPA

3. Оцінка продуктивності



,

$$p_n = \gamma_n T_n,$$

$$T_n = \frac{C_n}{T},$$

де a - один зі станів, у якому перебуває додаток,

pn - ймовірність перебування системи в стані *n*,

ϒn - кількість елементів/станів додатку

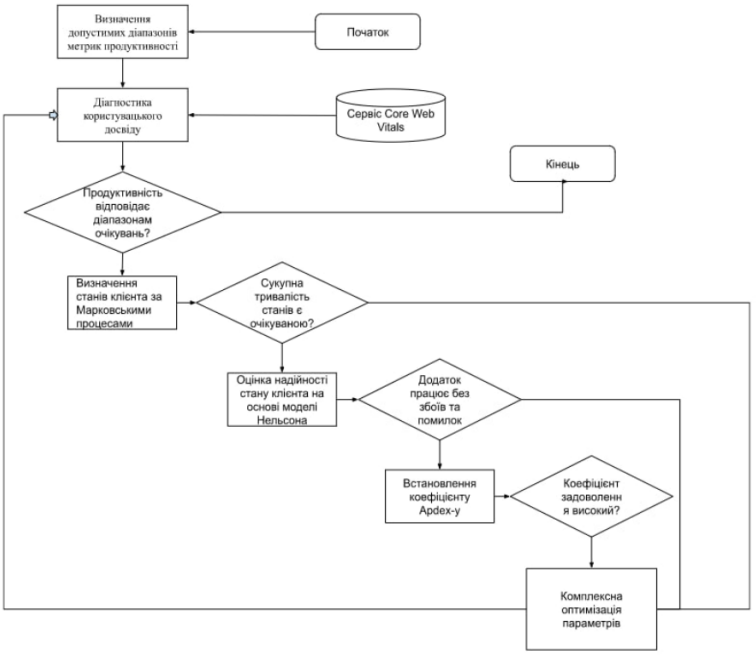
Tn - середній час перебування в системі у стані *n*.

Cn ймовірність переходу зі *n* стану в інший стан

T - середній час перебування в системі взагалі

8

УЗАГАЛЬНЕНИЙ АЛГОРИТМ ОПТИМІЗАЦІЇ

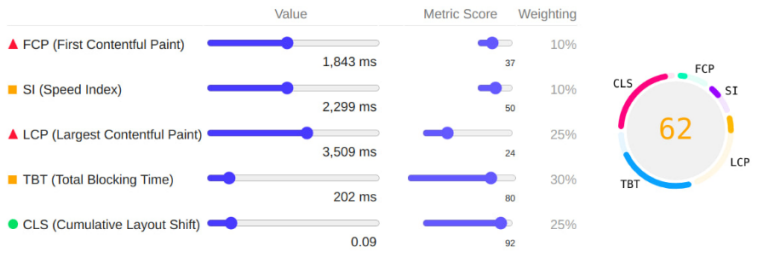


ОСНОВНІ СКЛАДОВІ ОПТИМІЗАЦІЇ

№	Категорія	Стратегія Оптимізації
1	Оптимізація програмного коду	Поліпшення процесу виявлення змін під час порівняння DOM структури
		Застосування бібліотеко-орієнтованих рішень
		Наслідування кращих практик розробки
2	Оптимізація комп’ютерної інфраструктури	Використання CDN
		Аналіз споживання ресурсів веб-сервера та уникнення перевантажень завдяки Load Balancer
		Правильне інфраструктурне масштабування
3	Оптимізація медіа додатків	Підбір підходящих форматів даних
		Мініфікація медіа-даних
		Компресія
4	Оптимізація мережі	Лініве завантаження модулів
		Використання більш просунутих стандартів - HTTP/3
		Кешування

ОЦІНКА ЕФЕКТИВНОСТІ ОПТИМІЗАЦІЇ

Angular додаток до оптимізації:

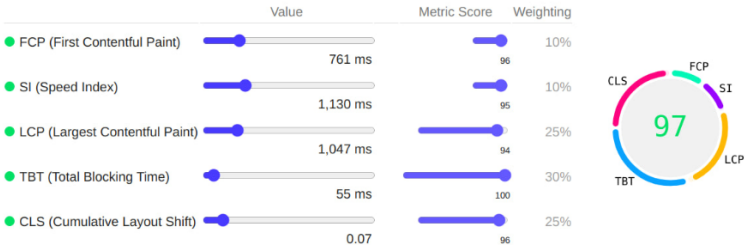


React додаток до оптимізації:

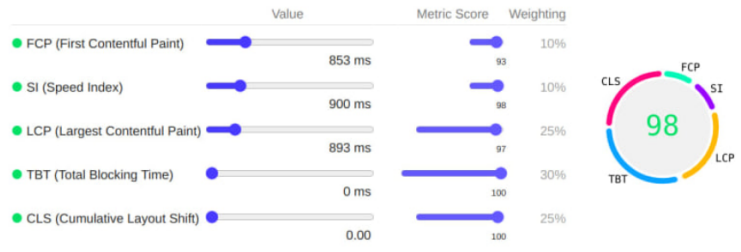
11

ОЦІНКА ЕФЕКТИВНОСТІ ОПТИМІЗАЦІЇ

Angular додаток після оптимізації:



React додаток після оптимізації:



12

ВИСНОВКИ

1. Проведено комплексне порівняння технологій, що використовують техніку часткового оновлення вмісту веб-сторінки.
2. Розглянуто методи впливу на параметри клієнтської частини веб-додатків, від яких напряму залежить підтримка високого рівня продуктивності.
3. Сформовано алгоритм оптимізації на основі математичних моделей категорій надійності, продуктивності, якості відображення, пріоритезуючи параметри відповідно до обсягу їх впливу на загальний рівень функціонування додатку.
4. Оптимізовано часткове оновлення сторінки клієнтських веб-додатків на основі платформ розробки Angular та React.js.
5. Проведено оцінку ефективності застосування розробленого методу вирішення проблеми часткового оновлення вмісту сторінки завдяки використанню відповідного сервісу Pagespeed Insights на основі Core Web Vitals.

13

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Статті:

1. Bondarchuk A. Khlysta I. Mitigating partial content update issues on web pages through SPA parameter optimization. State University of Information and Telecommunication Technologies. “Telecommunication and Information Technologies” 2023. [3]. (Подано до друку)

Тези доповідей:

1. Хлиста І.А., Бондарчук А.П. Вирішення проблеми часткового оновлення вмісту веб-сторінки шляхом оптимізації параметрів SPA V Всеукраїнська науково-практична конференція студентів, аспірантів та молодих вчених “Комп’ютерне моделювання та інформаційні технології” – Львів: НЛТУ, 2023. С. 286 - 291

14

ДЯКУЮ ЗА УВАГУ!

ДОДАТОК А

ФРАГМЕНТ КОДУ ДОДАТКУ ANGULAR ДО ОПТИМІЗАЦІЇ

```
import { inject, NgModule } from "@angular/core";
import { Routes, RouterModule, PreloadAllModules } from "@angular/router";
import { UserService } from "./core/services/user.service";
import { map } from "rxjs/operators";
import { ProfileComponent } from "./features/profile/profile.component";

const routes: Routes = [
 {
   path: "",
   loadComponent: () =>
     import("./features/home/home.component").then((m) => m.HomeComponent),
 },
 {
   path: "login",
   loadComponent: () =>
     import("./core/auth/auth.component").then((m) => m.AuthComponent),
   canActivate: [
     () => inject(UserService).isAuthenticated.pipe(map((isAuth) => !isAuth)),
   ],
 },
 {
   path: "register",
   loadComponent: () =>
     import("./core/auth/auth.component").then((m) => m.AuthComponent),
   canActivate: [
     () => inject(UserService).isAuthenticated.pipe(map((isAuth) => !isAuth)),
   ],
 },
 {
   path: "settings",
   loadComponent: () =>
     import("./features/settings/settings.component").then(
       (m) => m.SettingsComponent
     ),
   canActivate: [() => inject(UserService).isAuthenticated],
 },
 {
   path: "profile",
   children: [
     {
       path: ":username",
       component: ProfileComponent,
```

```
      children: [
        {
          path: "",
          loadComponent: () =>
            import("./features/profile/profile-articles.component").then(
              (m) => m.ProfileArticlesComponent
            ),
        },
        {
          path: "favorites",
          loadComponent: () =>
            import("./features/profile/profile-favorites.component").then(
              (m) => m.ProfileFavoritesComponent
            ),
        },
      ],
    },
  ],
},
{
  path: "editor",
  children: [
    {
      path: "",
      loadComponent: () =>
        import("./features/editor/editor.component").then(
          (m) => m.EditorComponent
        ),
      canActivate: [() => inject(UserService).isAuthenticated],
    },
    {
      path: ":slug",
      loadComponent: () =>
        import("./features/editor/editor.component").then(
          (m) => m.EditorComponent
        ),
      canActivate: [() => inject(UserService).isAuthenticated],
    },
  ],
},
{
  path: "article/:slug",
  loadComponent: () =>
    import("./features/article/article.component").then(
      (m) => m.ArticleComponent
```

```
    ),
  },
];

@NgModule({
  imports: [
    RouterModule.forRoot(routes, {
      preloadingStrategy: PreloadAllModules,
    }),
  ],
  exports: [RouterModule],
})
export class AppRoutingModule {}
```

ДОДАТОК Б

ФРАГМЕНТ КОДУ ДОДАТКУ REACT.JS ДО ОПТИМІЗАЦІЇ

```
import agent from '../agent';
import Header from './Header';
import React from 'react';
import { connect } from 'react-redux';
import { APP_LOAD, REDIRECT } from '../constants/actionTypes';
import { Route, Switch } from 'react-router-dom';
import Article from '../components/Article';
import Editor from '../components/Editor';
import Home from '../components/Home';
import Login from '../components/Login';
import Profile from '../components/Profile';
import ProfileFavorites from '../components/ProfileFavorites';
import Register from '../components/Register';
import Settings from '../components/Settings';
import { store } from '../store';
import { push } from 'react-router-redux';

const mapStateToProps = state => {
 return {
   appLoaded: state.common.appLoaded,
   appName: state.common.appName,
   currentUser: state.common.currentUser,
   redirectTo: state.common.redirectTo
 }};

const mapDispatchToProps = dispatch => ({
 onLoad: (payload, token) =>
   dispatch({ type: APP_LOAD, payload, token, skipTracking: true }),
 onRedirect: () =>
   dispatch({ type: REDIRECT })
});

class App extends React.Component {
 componentWillReceiveProps(nextProps) {
   if (nextProps.redirectTo) {
     // this.context.router.replace(nextProps.redirectTo);
     store.dispatch(push(nextProps.redirectTo));
     this.props.onRedirect();
   }
 }

 componentWillMount() {
```

```
    const token = window.localStorage.getItem('jwt');
    if (token) {
      agent.setToken(token);
    }

    this.props.onLoad(token ? agent.Auth.current() : null, token);
  }

  render() {
    if (this.props.appLoaded) {
      return (
        <div>
          <Header
            appName={this.props.appName}
            currentUser={this.props.currentUser} />
            <Switch>
            <Route exact path="/" component={Home}/>
            <Route path="/login" component={Login} />
            <Route path="/register" component={Register} />
            <Route path="/editor/:slug" component={Editor} />
            <Route path="/editor" component={Editor} />
            <Route path="/article/:id" component={Article} />
            <Route path="/settings" component={Settings} />
            <Route path="/@:username/favorites" component={ProfileFavorites} />
            <Route path="/@:username" component={Profile} />
            </Switch>
        </div>
      );
    }
    return (
      <div>
        <Header
          appName={this.props.appName}
          currentUser={this.props.currentUser} />
      </div>
    );
  }
}

export default connect(mapStateToProps, mapDispatchToProps)(App);
```

ДОДАТОК В

ФРАГМЕНТ КОДУ ОПТИМІЗОВАНОГО ДОДАТКУ ANGULAR

```
@UntilDestroy()
@Component({
 selector: "app-dashboard-page",
 templateUrl: "./dashboard.component.html",
 styleUrls: ["./dashboard.component.css"],
 changeDetection: ChangeDetectionStrategy.OnPush,
 imports: [
   NgClass,
   ArticleListComponent,
   AsyncPipe,
   LetDirective,
   NgForOf,
   ShowAuthedDirective,
 ],
 standalone: true,
})
export class DashboardComponent implements OnInit {
 isUserLoggedIn = false;

 serverSettingsConfig: ArticleListConfig = {
   type: "all",
   filters: {},
 };

 tags$ = inject(TagsService)
   .getAll()
   .pipe(tap(() => (this.tagsLoaded = true)));
 tagsLoaded = false;

 constructor(
   private readonly router: Router,
   private readonly userService: UserService
 ) {
 }

 ngOnInit(): void {
   this.userService.isAuthenticated
     .pipe(
       untilDestroyed(this),
       tap((isAuthenticated) => {
         if (isAuthenticated) {
           this.setListTo("feed");
```

```
        } else {
          this.setListTo("all");
        }
      }),
    )
    .subscribe(
      (isAuthenticated: boolean) => (this.isUserLoggedIn = isAuthenticated)
    );
 }

 setListTo(type: string = "", filters: Object = {}): void {
   if (type === "feed" && !this.isUserLoggedIn) {
     void this.router.navigate(["/login"]);
     return;
   }

   this.serverSettingsConfig = {type: type, filters: filters};
 }
}
```

```
<app-article-preview *ngFor="let article of (currentArticles$ | async)"
[article]="article">
</app-article-preview>

<p-progressSpinner *ngIf="loading ===
LoadingState.LOADING"></p-progressSpinner>

<div
 class="article-preview"
 *ngIf="loading === LoadingState.LOADED && !(currentArticles$ |
async)?.length"
>
 No articles are here... yet.
</div>

<app-article-pagination
 *ngIf="loading === LoadingState.LOADED && totalPages.length > 1"
 [totalPages]="totalPages"
 (evokeRequestingData)="acquireDataOnPageChange($event)"

></app-article-pagination>
```

```
@UntilDestroy()
@Component({
 selector: "app-favorite-button",
 templateUrl: "./favorite-button.component.html",
 changeDetection: ChangeDetectionStrategy.OnPush,
 imports: [NgClass],
 standalone: true,
})
export class FavoriteButtonComponent {
 @Input() article!: Article;
 @Output() toggle = new EventEmitter<boolean>();

 isSubmitting = false;

 constructor(
   private readonly articleService: ArticlesService,
   private readonly router: Router,
   private readonly userService: UserService
 ) {
 }

 toggleFavorite(): void {
   this.isSubmitting = true;

   this.userService.isAuthenticated
     .pipe(
       untilDestroyed(this),
       switchMap((authenticated) => {
         if (!authenticated) {
           void this.router.navigate(["/login"]);
           return EMPTY;
         }

         if (!this.article.favorited) {
           return this.articleService.favorite(this.article.slug);
         } else {
           return this.articleService.unfavorite(this.article.slug);
         }
       })
     )
     .subscribe({
       next: () => {
         this.isSubmitting = false;
         this.toggle.emit(!this.article.favorited);
       },
```

```
        error: () => (this.isSubmitting = false),
      });
  }
}
```

ДОДАТОК Г

ФРАГМЕНТ КОДУ ОПТИМІЗОВАНОГО ДОДАТКУ REACT.JS

```
import React from 'react';
import agent from '../../agent';

const Tags = props => {
 const tags = props.tags;
 if (tags) {
   return (
     <div className="tag-list">
       {
         tags.map(tag => {
           const handleClick = ev => {
             ev.preventDefault();
             props.onClickTag(tag, page => agent.Articles.byTag(tag, page),
agent.Articles.byTag(tag));
           };

           return (
             <a
               href=""
               className="tag-default tag-pill"
               key={tag}
               onClick={handleClick}>
               {tag}
             </a>
           );
         })
       }
     </div>
   );
 } else {
   return (
     <div>Loading Tags...</div>
   );
 }
};

export default Tags;
```