УДК 004.4:004.77

ВПЛИВ СТРАТЕГІЇ РЕНДЕРИНГУ (ISR, RSC) НА ПРОДУКТИВНІСТЬ NEXT.JS-ЗАСТОСУНКІВ: ЕКСПЕРИМЕНТАЛЬНА ОЦІНКА

Вишневський Є.В.[1]; Немченко В.В., к.т.н., доц.[2]
[1,2] Черкаський державний технологічний університет
[1,2] Україна, Черкаси
[1] evgenii.vishnevsk@gmail.com; [2] v.nemchenko@chdtu.edu.ua

Анотація. *Досліджено вплив стратегій рендерингу Next.js (CSR, ISR, SSR) та React Server Components на метрики продуктивності вебзастосунків. Запропоновано критерій вибору стратегії за типом контенту й методику вимірювання з реальним дроселюванням. Експериментально встановлено зниження Total Blocking Time на 61 % та зростання Performance Score на 24 бали за статистично незначущої зміни TTFB, FCP і LCP.*

Ключові слова: *рендеринг; Next.js; React Server Components; інкрементальна статична регенерація; Core Web Vitals.*

Вступ. Сучасні вебзастосунки еволюціонували від односторінкових (SPA) до гібридних систем, де швидкість завантаження й інтерактивності безпосередньо визначає якість програмного забезпечення та ефективність пошукової оптимізації. Стандартом оцінювання стали метрики Core Web Vitals [1] і пов'язані з ними показники TTFB, FCP, LCP та TBT. Фреймворк Next.js пропонує клієнтський (CSR) і серверний (SSR) рендеринг, статичну генерацію (SSG), інкрементальну статичну регенерацію (ISR), а також архітектуру React Server Components (RSC), що переносить виконання частини компонентів на сервер [2; 3]. Поширеним є припущення, що перехід на ISR/RSC універсально прискорює всі метрики, проте воно потребує експериментальної перевірки.

Мета роботи – обґрунтувати вибір стратегії рендерингу залежно від типу контенту та експериментально оцінити вплив ISR і RSC на метрики продуктивності Next.js-застосунку.

Основна частина. Продуктивність подано вектором $P = \langle TTFB, T^{FCP}, T^{LCP}, T^{TBT}, S^{bundle} \rangle$, де S^{bundle} – розмір початкового JavaScript-бандла. Вибір стратегії формалізовано як мінімізацію зваженої суми:

$$F = \omega^1 \cdot TTFB + \omega^2 \cdot T^{FCP} + \omega^3 \cdot T^{LCP} + \omega^4 \cdot T^{TBT} + \omega^5 \cdot S^{bundle} \to \min, \quad (1)$$

де ω_i – вагові коефіцієнти, що відображають пріоритети конкретного проєкту. З (1) випливають правила: для рідко змінюваного контенту доцільна SSG; для каталогів помірної волатильності – ISR; для даних, що мають бути актуальними на кожен запит, – SSR; для персоналізованого стану (кошик користувача) – CSR.

Експериментальне середовище – демонстраційний вебзастосунок інтернет-магазину меблів (Next.js App Router, TypeScript, SQLite). Порівнювали дві конфігурації: неоптимізовану (CSR: дані каталогу завантажуються на клієнті після гідратації) та оптимізовану (ISR + RSC для каталогу, SSR для сторінки товару, CSR для кошика). Вимірювання виконано програмним API Lighthouse [4] у headless Chrome з реальним (не симульованим) дроселюванням мережі (RTT 40 мс, ≈4 Мбіт/с) і процесора (4x); N = 7 прогонів на конфігурацію, прогони чергувалися (A/B/A/B), як оцінку використано медіану. Реальне дроселювання обрано тому, що симульоване (Lantern) некоректно враховує штучні серверні затримки [4]. Час блокування головного потоку (TBT) оцінювали за [5].

Результати наведено в табл. 1. Показники TTFB, FCP та LCP істотно не змінилися: на досліджуваній сторінці каталогу вони визначаються часом завантаження зображень під

мережевим дроселюванням, а інтерактивна сітка товарів із фільтром лишається клієнтським компонентом в обох конфігураціях. Натомість TBT скоротився з 70 до 27 мс (−61 %), а Performance Score зріс із 71 до 95 балів. Причина – усунення циклу клієнтського отримання даних після гідратації: оптимізована конфігурація отримує готовий HTML і виконує гідратацію один раз [2; 3].

Таблиця 1 – Медіанні метрики Lighthouse (N = 7)

Метрика	**CSR (неоптимізована)**	**ISR + RSC (оптимізована)**	**Зміна**
TTFB	15 мс	15 мс	без змін
FCP	2346 мс	2339 мс	без змін
LCP	2346 мс	2429 мс	+4 % (у межах шуму)
TBT	70 мс	27 мс	−61 %
Speed Index	2518 мс	2470 мс	−2 %
Performance Score	71 / 100	95 / 100	+24 бали

Реальне дроселювання та контроль паритету функціональності конфігурацій усунули дві методологічні похибки, виявлені в ході експерименту.

Висновки. 1. Запропоновано критерій (1) і правила вибору стратегії рендерингу за типом контенту. 2. Ефект ISR/RSC залежить від метрики та типу сторінки: встановлено значуще зниження TBT на 61 % і зростання Performance Score на 24 бали, але не виявлено змін TTFB, FCP та LCP для сторінки з домінуючим медіаконтентом, що уточнює припущення про універсальне прискорення Core Web Vitals. 3. Методика вимірювання (реальне дроселювання, чергування прогонів, контроль паритету функціональності) придатна для оцінювання e-commerce-платформ. Перспективи – сторінки без домінуючого медіаконтенту та реальне зовнішнє джерело даних.

Використання інструментів ШІ. Claude (Anthropic, модель Claude Sonnet 5.5) використано для підготовки й редагування тексту розділів «Вступ», «Основна частина» та «Висновки» на основі матеріалів кваліфікаційної роботи авторів і для оформлення тез за вимогами шаблону. Результати експерименту, їх інтерпретацію та остаточну редакцію перевірено авторами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Google. (n.d.). *Core Web Vitals*. web.dev. Retrieved from https://web.dev/articles/vitals.

[2] Vercel. (n.d.). *Next.js documentation: Rendering*. Retrieved from https://nextjs.org/docs/pages/building-your-application/rendering.

[3] Meta Platforms. (n.d.). *Server Components*. React documentation. Retrieved from https://react.dev/reference/rsc/server-components.

[4] Google Chrome Team. (n.d.). *Lighthouse throttling*. GitHub. Retrieved from https://github.com/GoogleChrome/lighthouse/blob/main/docs/throttling.md.

[5] Google. (n.d.). *Total Blocking Time (TBT)*. web.dev. Retrieved from https://web.dev/articles/tbt.

Vyshnevskyi Ye.V., Nemchenko V.V.

Impact of rendering strategy (ISR, RSC) on the performance of Next.js applications: an experimental evaluation

Abstract. *The effect of Next.js rendering strategies (CSR, ISR, SSR) and React Server Components on web application performance metrics is studied. A criterion for choosing a strategy by content type and a measurement method with real throttling are proposed. Total Blocking Time is experimentally shown to decrease by 61 % and the Lighthouse Performance Score to rise by 24 points, while TTFB, FCP and LCP change insignificantly.*

Keywords: *rendering; Next.js; React Server Components; incremental static regeneration; Core Web Vitals.*