

ПЕРСПЕКТИВИ ВИКОРИСТАННЯ СЕРВЕРНОГО РЕНДЕРИНГУ У КОНТЕКСТІ РОЗРОБКИ WEB-РЕСУРСУ «ОНЛАЙН ФОРУМ»

Вінницький національний технічний університет

Анотація

Розглянуто перспективи використання серверного рендерингу (SSR) під час розробки WEB-ресурсу «Онлайн форум». Увагу приділено перевагам серверного підходу в порівнянні з клієнтським рендерингом, а саме, покращення продуктивності, оптимізації SEO, швидшого завантаження сторінок та зручності індексації контенту. Досліджено також виклики та можливості, пов'язані з реалізацією SSR у сучасних WEB-застосунках.

Ключові слова: серверний рендеринг, SSR, онлайн форум, оптимізація продуктивності, SEO, динамічний контент, індексація контенту, WEB-застосунок, серверні розрахунки, кешування контенту, швидкодія застосунків, фронтенд, WEB-ресурс, односторінковий застосунок, SPA, TTI.

Abstract

The prospects of using server rendering (SSR) during the development of the WEB resource "Online Forum" are considered. Attention is paid to the advantages of a server-side approach compared to client-side rendering, namely, improved performance, SEO optimization, faster page loading and ease of content indexing. The challenges and opportunities associated with the implementation of SSR in modern WEB applications were also studied.

Keywords: server rendering, SSR, online forum, performance optimization, SEO, dynamic content, content indexing, WEB application, server calculations, content caching, application speed, frontend, WEB resource, single page application, SPA, TTI.

Вступ

Відповідно до результатів опитування розробників на ресурсі State of JavaScript 2024, на сьогодні серверний рендеринг стає надзвичайно популярним у розробці WEB-застосунків. Це пов'язано зі зростанням вимог до швидкості завантаження сайтів, оптимізації пошукових систем (SEO) та користувацького досвіду. З іншого боку стрімкий розвиток фреймворків, які підтримують гібридні підходи до створення сторінок, змушує розробників розробляти нові підходи до архітектури сучасних WEB-рішень. Саме тому нині питання вибору ефективного способу рендерингу набуває особливої актуальності [1].

Метою цього дослідження є визначення факторів популярності серверного рендерингу, його значення у створенні продуктивних та зручних WEB-застосунків, а також оцінювання переваг та недоліків цього підходу у контексті розробки онлайн форумів. У результаті буде проаналізовано в яких випадках серверний рендеринг забезпечує найкращі результати порівняно з клієнтським або змішаним підходами.

Результати дослідження

Натепер більшість бібліотек та фреймворків, наприклад, React, Vue та Angular, не рекомендують використовувати їх у чистому вигляді для реалізації серверного рендерингу. Натомість вони рекомендують обирати деякий мета-фреймворк, наприклад, Next, Nuxt тощо. Однією із причин є те, що реалізація серверних компонентів, які з'явилися в одних із останніх версій вищенаведених бібліотек та фреймворків, суттєво залежать від обраного мета-фреймворку. У документації React, Vue, Angular досить мало інформації про серверний рендеринг, оскільки його реалізація залежить від обраного мета-фреймворку. Наприклад, Next уже підтримує серверні компоненти та має належну документацію, де можна дізнатися, як ними користуватися. Також його перевагою є те, що він додає лише найнеобхідніше та найзручніше, і у невеликих обсягах диктує свої вимоги до побудови застосунків, саме тому він досить наближений до React, що за умови наявності базових знань, дозволить просто

освоїти Next. Якщо це буде інший фреймворк, то все працюватиме по-іншому. Саме тому розробники радять використовувати більш високорівневі інструменти [2].

Клієнтський рендеринг (CSR) – це за замовчуванням стандарт роботи Single Page Application (SPA). Цей стандарт раніше використовувався у всіх бібліотеках та фреймворках, тобто рендеринг відбувався лише на стороні клієнта, а саме, у браузері користувача. Однак з часом виникла потреба у більш швидших підходах до рендерингу сторінок, що дозволило б пришвидшити завантаження контенту та покращити SEO. Саме тому було розроблено такі методи серверного рендерингу як SSR, SSG та ISR [3].

Server-side Rendering (SSR) – це алгоритм роботи застосунку, коли HTML сторінка генерується на кожний запит користувача. Цей принцип роботи досить схожий на роботу з PHP або Python. Наприклад, існує платформа для розробки WEB-форумів XenForo, яка побудована на PHP, тобто на кожен запит користувача буде надано згенеровану сторінку, саме тому при масштабуванні застосунків можливі проблеми з продуктивністю, оскільки не на кожен запит доцільно проводити повторну генерацію сторінки [4].

Static site Generation (SSG) – це вид серверного рендерингу, коли усі потрібні HTML сторінки генеруються під час збірки застосунку і використовуються повторно для кожного запиту. Тобто, такі сторінки зберігаються на сервері і за потреби повертаються усім користувачам [4].

Incremental Static Regeneration (ISR) – це вид серверного рендерингу, схожий за своєю природою на SSG, але дозволяє за таймером чи подією оновлювати статичні сторінки після завершення збірки сайту. Це дозволяє використовувати статичну генерацію для окремих сторінок, які досить зрідка оновлюються, без необхідності перебудовувати весь сайт. При такому підході оновлюються лише ті сторінки, які змінилися [4].

У сучасних мета-фреймворках за замовчуванням попередньо використовується саме SSG. Це означає, що генерується HTML для кожної сторінки заздалегідь, замість того щоб виконувати весь рендеринг на стороні клієнта через JavaScript. Цей підхід дозволяє підвищити продуктивність і покращити SEO. Кожна сторінка, яка була згенерована містить мінімальний обсяг JavaScript коду. Коли браузер завантажує сторінку, то відбувається процедура гідратації, тобто під час отримання статичного сайту, відправленого сервером, сайт наповнюється JavaScript кодом та стає динамічним сайтом, який може реагувати на зміну даних сайту, тобто перетворюється у SPA [5].

Рекомендується використовувати саме SSG, коли це можливо. Проте, коли необхідно відображати динамічні дані залежно від запиту користувача, доцільно розглянути підхід з SSR чи навіть CSR [6].

Навіть на сьогодні підхід до серверного рендерингу поступово змінюється. Замість прямого поділу на SSR та SSG тепер використовують статичні, динамічні рендеринги та стрімінг. Статичний рендеринг використовується за замовчуванням та працює аналогічно SSG. Динамічний рендеринг працює як SSR, який досить корисний, коли сторінка містить дані, які не є однаковими для кожного користувача, тобто містить інформацію, яка може бути відома лише під час запиту. Стрімінгом називають той процес, коли інформація передається користувачу поступово, тобто ця інформація розбивається на фрагменти, які надаються клієнту по готовності [7].

Якщо фреймворк не може використати статичну генерацію, він автоматично перемикається на серверний рендеринг. Цей підхід дозволяє розділяти компоненти на серверні компоненти, які запускаються один раз на запит на сервері, тобто коли приходить запит, компонент отримує дані та повертає результат. Код серверного компонента на клієнт ніколи не потрапляє, проводячи аналогію з інструментом XenForo вся логіка відбувається на сервері і зрештою клієнту надходить лише результат. У такого виду компонентів немає власного стану та етапів життєвого циклу, тому тут не можна використовувати сторонні API. Завдяки такому простому алгоритму серверні компоненти працюють значно ефективніше. При їх використанні, якщо перевірити мережеві запити чи перезавантажити сторінку, можна побачити, що відсутні fetch-запити. Це пояснює швидке завантаження сторінок, оскільки дані вже присутні на сторінці, вона повністю готова та просто передається сервером клієнту. Якщо б запит відбувався з браузера, процес був би повільнішим. Замість миттєвого відображення сторінки спершу надходив би запит, сервер формував би сторінку та повертав її користувачу. Це значно довше порівняно зі статичною генерацією [7, 8].

Серверні обчислення дозволяють користувачу отримувати файли меншого розміру, що зменшує навантаження на його пристрій шляхом зменшення обсягу даних і часу обробки. Під час розробки онлайн форумів, де можуть бути наявні великі обсяги даних або зображень, така оптимізація значно покращує швидкість завантаження сторінок та зменшує затримки. Використання серверного підходу

дозволяє уникнути додаткових запитів користувачів до сервісів з потрібною інформацією. З таким підходом користувачу не потрібно чекати додаткових завантажень для доступу до важливої інформації, такої як профілі, пости чи коментарі. Через надходження серверно згенерованої сторінки, зникає проблема поганої індексації застосунку, оскільки уся необхідна SEO інформація надходить від сервера під час пре-рендеру навіть до моменту гідратації. Це означає, що пошукові системи можуть миттєво індексувати вміст сторінки. Під час первинного завантаження сторінки без серверного рендерингу, тобто SPA, браузер отримує великий обсяг ресурсів, включаючи HTML, CSS, JavaScript, картинки тощо, і потребує часу на їх обробку та виконання. Це може призвести до значних затримок у порівнянні з серверно згенерованими сторінками, де браузер отримує вже готовий HTML-контент. Для онлайн форумів цей аспект є важливим, оскільки затримка при завантаженні може бути критичною, особливо якщо форум містить великий обсяг контенту або медіа [9].

Завдяки серверному рендерингу, WEB-застосунки отримують значну перевагу у вигляді прямого доступу до бази даних та серверної логіки під час генерування сторінки. Це дозволяє серверу динамічно створювати контент, який може включати актуальну інформацію з бази даних, наприклад, при кожному завантаженні сторінки сервер може відразу передати користувачеві найновіші пости, коментарі та статистику активності. Серверні компоненти доцільно використовувати також, коли потрібно безпосередньо отримати або обробити дані, наприклад, запит на пости, коментарі чи профілі користувачів. Коли на сервері зберігаються чутливі дані, такі як ключі доступу, токени безпеки або будь-яка інша конфіденційна інформація, серверні компоненти дозволяють ізолювати ці дані від клієнтської частини, що значно підвищує безпеку застосунку. Зберігання важливих або великих залежностей на сервері дозволяє зменшити обсяг JavaScript, який потрібно завантажити та виконати на клієнтському пристрої [10].

Хоча користувач відразу бачить серверно згенеровану сторінку з необхідним контентом, через процес гідратації та пре-рендерингу інтерфейс не стає повністю інтерактивним відразу. Це може вплинути на показник «Time to Interactive» (TTI), який важливий для швидкості взаємодії з сайтом. Користувачі хоча і отримують візуальний контент миттєво, не можуть негайно взаємодіяти з елементами сторінки, такими як кнопки, форми або пости. Для онлайн-форумів це може бути проблемою, оскільки активні користувачі очікують швидкої реакції на їхні дії, наприклад, коментування чи перегляд нових повідомлень. При використанні серверного рендерингу весь процес генерації сторінки відбувається на сервері, що створює додаткове навантаження, оскільки кожен запит потребує обробки, виконання логіки, отримання даних з бази та рендерингу контенту. У разі високої активності на форумі, коли кілька користувачів одночасно взаємодіють з платформою, це може призвести до значних затримок або навіть до перевантаження сервера [5, 7, 9, 10].

Отже, використання серверного рендерингу є доцільним, коли дані отримуються безпосередньо на сервері; потрібен прямий доступ до бекенду та його утиліт; потрібно мінімізувати ймовірність викрадення чи спотворення важливих або критично важливих даних, шляхом їх зберігання на сервері недоступному для клієнта. Цей підхід також доцільно застосовувати для зниження навантаження на клієнтську частину шляхом передачі частини завдань серверу, а також у випадках, коли необхідно забезпечити швидке завантаження контенту для поліпшення SEO. Однак серверний рендеринг не є ефективним у випадках, коли потрібно обробляти події користувача (event), працювати з DOM API або використовувати локальний стан застосунку. Також цей підхід менш доцільний для застосунків із динамічною та складною взаємодією з користувачем, яка потребує частого перерендерингу контенту на стороні клієнта.

Висновки

Відповідно до проведеного дослідження встановлено, що серверний рендеринг є важливим інструментом для покращення швидкості завантаження і SEO-оптимізації онлайн форумів, оскільки зменшує час до першого рендерингу і забезпечує кращу індексацію контенту. Однак для високої інтерактивності, яку потребують сучасні WEB-застосунки, цей метод має свої обмеження. Проблеми виникають, коли потрібно інтегрувати такі функціональності, як використання DOM API, хуків чи методів життєвого циклу компонентів, браузерних API та обробки подій на умовах чи івентах, що є необхідними для динамічних взаємодій. Це обмежує можливості інтерактивності на серверно згенерованих сторінках, оскільки процес гідратації та переробка інтерфейсу після завантаження може спричинити затримки. Для онлайн форумів, де важливою є швидка реакція на дії користувачів, наприклад, коментування чи оновлення постів, ці затримки можуть бути критичними. Таким чином,

зважаючи на те, що серверний рендеринг має суттєві переваги в контексті швидкості та SEO, він не завжди є оптимальним для створення застосунків, що потребують високої інтерактивності. Враховуючи цей фактор потрібно чітко продумати, які компоненти мають рендеритись на сервері, а які – на клієнті, для досягнення оптимального балансу між продуктивністю та взаємодією з користувачем.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Libraries Experience & Sentiment. URL: <https://2024.stateofjs.com/en-US/libraries/> (дата звернення: 29.01.2025).
2. Start a New React Project. URL: <https://react.dev/learn/start-a-new-react-project> (дата звернення: 29.01.2025).
3. Client-side Rendering. URL: <https://www.patterns.dev/react/client-side-rendering/> (дата звернення: 30.01.2025).
4. Server-side Rendering (SSR). URL: <https://nextjs.org/docs/pages/building-your-application/rendering/server-side-rendering> (дата звернення: 30.01.2025).
5. Client Hydration. URL: <https://vuejs.org/guide/scaling-up/ssr.html#client-hydration> (дата звернення: 30.01.2025).
6. Rendering. URL: <https://nextjs.org/docs/pages/building-your-application/rendering> (дата звернення: 30.01.2025).
7. Чому React Server Components – майбутнє веброзробки. URL: <https://dou.ua/forums/topic/46138/> (дата звернення: 30.01.2025).
8. Server Components. URL: <https://react.dev/reference/rsc/server-components> (дата звернення: 30.01.2025).
9. Порівнюємо способи генерації сторінок: CSR, SSR, SSG, ISR. Гайд на основі стеку React. URL: <https://dou.ua/forums/topic/41585/> (дата звернення: 30.01.2025).
10. A Deep Dive into Server Side Rendering (SSR). URL: <https://www.linkedin.com/pulse/deep-dive-server-side-rendering-ssr-ashok-vishwakarma-w7jpc> (дата звернення: 30.01.2025).

Попович Юрій Юрійович – студент групи 2КН-21б, факультет інтелектуальних інформаційних технологій та автоматизації, Вінницький національний технічний університет, м. Вінниця, email: upopovic291@gmail.com.

Крилик Людмила Вікторівна – к.т.н, доцент, доцент кафедри комп'ютерних наук, Вінницький національний технічний університет, м. Вінниця.

Popovich Yuriy Y. – Faculty of Intelligent Information Technology and Automation, Vinnytsia National Technical University, Vinnytsia, email: upopovic291@gmail.com.

Krylik Lyudmila V. – PhD (Eng.), Associate Professor of Department for Computer Science, Vinnytsia National Technical University, Vinnytsia.