DOI: 10.25140/2411-5363-2024-4(38)-197-207
УДК 004.9

Кирило Олегович Насенок[1], Марія Михайлівна Войцеховська[2]

[1]аспірант, здобувач наукового ступеня доктор філософії за спеціальністю 122
Національний університет «Чернігівська політехніка» (Чернігів, Україна)
E-mail: kaboo@stu.cn.ua. **ORCID:** https://orcid.org/0009-0004-0972-7086

[2]доктор філософії, доцент кафедри інформаційних технологій і програмної інженерії
Національний університет «Чернігівська політехніка» (Чернігів, Україна)
E-mail: m.voitsekhovska@stu.cn.ua. **ORCID:** https://orcid.org/0000-0002-1711-101X

ПРОБЛЕМАТИКА КЛІЄНТСЬКОГО РЕНДЕРИНГУ В СУЧАСНІЙ ВСЕСВІТНІЙ МЕРЕЖІ

Стаття надає огляд сучасних проблем клієнтського рендерингу та їхній вплив на всесвітню мережу. Описані та згруповані основні проблеми, які з'являються при використанні клієнтського рендерингу. Зокрема, описано вплив клієнтського рендерингу на час завантаження та розмір вебсторінок, проблеми з пошуком через пошукові системи, а також вразливості, які існують у безпеці додатків із клієнтським рендерингом. Стаття включає аналіз можливостей вирішення цих проблем та оптимізації роботи всесвітньої павутини

Ключові слова: *клієнтський рендеринг; вебдодаток; клієнтський додаток; оптимізація; всесвітня мережа; безпека; пошукова система.*

Рис.: 2. Бібл.: 14.

Актуальність теми дослідження. Клієнтський рендеринг — це один із методів реалізації рендерингу клієнтських додатків, зокрема вебсторінок, які переглядаються у браузері. У 2013 році, коли цей підхід здобував популярність, він став революційним для розробників і всієї вебіндустрії, оскільки дозволяв створювати логічний та компонентний код, який легко розширюється і повторно використовується. На 2024 рік приблизно 9,5 мільйонів активних і високонавантажених сайтів використовують цей метод, що становить близько 8 % усіх активних сайтів і 17 % світового трафіку. Проте, як і кожна технологія, клієнтський рендеринг має свої недоліки.

Ці недоліки впливають на різні аспекти функціонування, починаючи від вразливості захисту додатка й закінчуючи проблемами з індексацією в пошукових системах. Проте найбільшою проблемою є збільшення обсягу файлів, що передаються через інтернет для завантаження та перегляду додатка. У масштабах звичайного застосунку це не є критичним, але для великого високонавантаженого проєкту, який щодня відвідують десятки мільйонів користувачів, це стає суттєвою проблемою, адже створює значне навантаження на мережу.

Постановка проблеми. На 2023 рік середній обсяг трафіку, що циркулює в Інтернеті, становить приблизно 150,7 ексабайта на місяць, що дорівнює 150 мільярдам гігабайт [1]. За останні п'ять років цей показник зріс на 21 %, і тенденція до зростання триває щороку. Інфраструктура інтернету є найбільшою та найскладнішою для обслуговування у світі. Вона постійно вдосконалюється: традиційні комунікації поступаються місцем оптоволокну, а пропускна спроможність мережі стрімко зростає. Проте, на жаль, це зростання відбувається не так швидко, як збільшення обсягів трафіку.

Кожен вебдодаток забезпечує багатий інтерактивний досвід, що сприяє зручній та ефективній комунікації з користувачами. Це охоплює функції, такі як заповнення форм, перегляд та завантаження даних, а також їх обробка і багато іншого. Протягом останнього десятиліття 82 % проєктів перейшли на фреймворки для клієнтських додатків, такі як Angular, React, Vue та інші, що значно поліпшило інтерактивність та спростило розробку. Цей перехід кардинально змінив підходи до створення вебдодатків, надаючи розробникам зручні інструменти для роботи, а користувачам — швидкі, зручні та якісні рішення.

Ціна інтерактивності визначається розміром додатка. Коли користувач вперше завантажує сторінку, він отримує невеликий HTML-файл разом із великим JavaScript-файлом, який відповідає за формування сторінки та забезпечує інтерактивні елементи [1]. Крім того, кожен запит до сервера, що надсилається додатком для обробки даних, також має свої розміри, що додатково навантажує мережу.

Таким чином, зручність розробки, підтримки та продуктивність додатка додає нові проблеми, такі як збільшення розміру файлу додатка, проблеми з пошуковими системами та проблеми з розподілом ресурсів.

Аналіз останніх публікацій у напрямку клієнтського рендерингу свідчить про занепокоєння наукової спільноти проблемою клієнтського рендерингу. Багато авторів описують та наводять приклади проблем із пошуковими системами, продуктивність на слабких пристроях, а також проблем з кешуванням та безпекою додатка.

Однією з основних проблем є надмірне завантаження світового трафіка, яке збільшується швидше за розвиток комунікаційних та апаратних приладів, які забезпечують роботу світової мережі [2].

Значна увага приділяється переліку практик та переваг, які надає клієнтський рендеринг, оскільки цей підхід дуже спрощує роботу розробників та робить добре підтримуваним та легко розширюваним. Використання підходу Pagelets, зручна навігація всередині проєкту та зниження навантаження на сервер робить підхід серверного рендерингу надзвичайно зручним та перспективним при виборі технології розробки [3].

Для оптимізації існуючого проєкту описані такі методики, як меморизація компонентів, оптимізація контексту та стану кожного компонента й відкладене завантаження компонентів [7]. Ці методики допомагають покращити рендеринг додатка та трохи зменшити розмір JavaScript файлу, проте не можуть виправити корінні проблеми підходу клієнтського рендерингу. Тому розглянуто інші методики рендерингу, які замінюють клієнтський рендеринг та позбавляють проєкт багатьох його проблем, таких як серверний рендеринг [3] та статична генерація [4]. Проте відсутній детальний опис проблеми клієнтського рендерингу, який зміг би продемонструвати, які технічні та архітектурні рішення накладають обмеження та спричиняють проблеми під час реалізації проєктів.

Виділення недосліджених частин загальної проблеми. Для ефективного вирішення цієї проблеми та розробки адекватних рішень необхідно глибоке розуміння витоків цієї ситуації, причин її виникнення, а також аналіз тих аспектів, зміна яких може сприяти загальному покращенню. Це передбачає всебічний опис проблем, пов’язаних із клієнтським рендерингом, детальний розбір принципів його роботи та вивчення реалізації модулів, що призводять до зниження продуктивності, уразливостей у безпеці та інших труднощів.

Важливо розглянути, як саме механізми клієнтського рендерингу впливають на ефективність завантаження та відображення вебсторінок. Дослідження цих процесів дозволить виявити ключові фактори, які спричиняють затримки в рендерингу, а також визначити можливі шляхи оптимізації. Серед важливих аспектів, які потребують уваги, можна виділити управління ресурсами, оптимізацію JavaScript-коду, а також покращення методів завантаження та кешування даних.

Це дозволить не тільки знизити навантаження на мережу, але й підвищити загальну продуктивність вебдодатків, а також забезпечити вищий рівень безпеки, що вкрай важливо в умовах зростаючих кіберзагроз. Зосередившись на цих питаннях, можна досягти комплексного вирішення проблеми клієнтського рендерингу, що позитивно вплине на користувацький досвід загалом.

Мета статті. Метою цієї статті є загальний опис проблематики клієнтського рендерингу в умовах сучасної всесвітньої мережі. А також перегляд альтернативних підходів, які вирішують частину проблем клієнтського рендерингу.

Виклад основного матеріалу.

Клієнтський рендеринг — це метод, за якого вебдодаток або його елементи формуються безпосередньо у браузері користувача, а не на сервері. Цей підхід здобув популярність завдяки таким фреймворкам, як React, Vue.js, Angular, Svelte та інші [5]. Він

забезпечує розробникам можливість створювати динамічні та інтерактивні вебдодатки, оскільки основна обробка відбувається на стороні клієнта. Це зменшує навантаження на сервер, адже він виконує лише роль API для надання даних. Завдяки цьому, розробка стає більш гнучкою та зручною, що дозволяє швидше реалізовувати нові функціональні можливості. Крім того, клієнтський рендеринг покращує користувацький досвід завдяки швидким переходам між сторінками без повторного завантаження.

Деталі реалізації в різних фреймворках можуть відрізнятися, проте основний принцип взаємодії клієнта в них однаковий:

1. Сервер отримує запит на отримання сторінки.
2. Сервер повертає на клієнт базовий HTML файл та JavaScript файл який має відобразити сторінку.
3. JavaScript файл будує та відображає сторінку в браузері.

На рис. 1 зображена схема роботи клієнтського рендерингу.

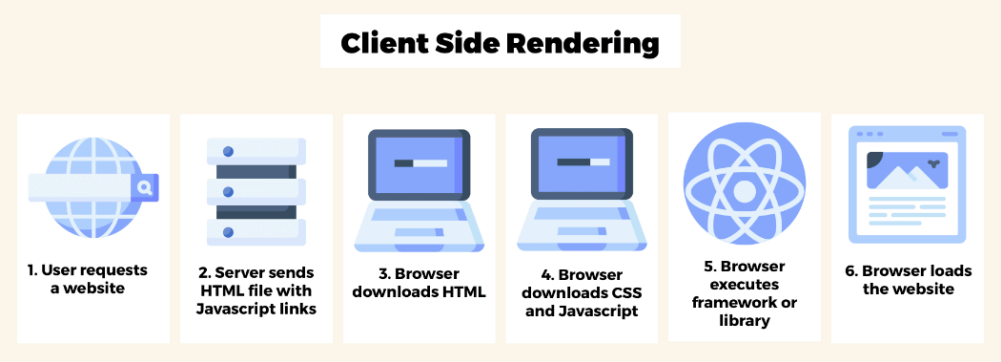

Рис. 1. Схема роботи клієнтського рендерингу

Джерело: розроблено авторами.

Однією з ключових концепцій, які стоять за клієнтським рендерингом, є використання Virtual DOM. DOM (Document Object Model) — це програмний інтерфейс, який представляє структуру HTML або XML документів. Багато сучасних фреймворків, таких як React, Angular і Vue.js, використовують Virtual DOM для оптимізації процесу рендерингу. Virtual DOM — це легка копія реального DOM, яка дозволяє фреймворку відстежувати зміни у стані програми без безпосередньої взаємодії з реальним DOM, що є дорогим процесом [6]. Коли стан програми змінюється, фреймворк оновлює Virtual DOM, а потім порівнює його з попередньою версією, визначаючи, які зміни потрібно внести до реального DOM. Це дозволяє зменшити кількість маніпуляцій з DOM і підвищити загальну продуктивність вебдодатка.

Клієнтський рендеринг дозволяє досягти високої інтерактивності вебдодатків, оскільки зміни в інтерфейсі можуть бути реалізовані без перезавантаження сторінки. Це досягається за рахунок використання AJAX-запитів або Fetch API для динамічного отримання даних з сервера. Коли користувач взаємодіє з елементами інтерфейсу, JavaScript виконує запити на сервер для отримання нових даних або оновлення існуючих, а потім рендерить ці дані на сторінці. Це дозволяє створювати односторінкові додатки (SPA), де користувач відчуває безшовний перехід між різними частинами програми.

Механізм роботи клієнтського рендерингу в браузері. Після того як браузер отримує необхідні файли, як показано на рис. 1, він формує базову структуру сторінки з HTML-документа. На цьому етапі створюється DOM-дерево без контенту, а також визначаються всі доступні стилі для побудови CSSOM (CSS Object Model) — структури, що представляє стилі для кожного елемента в DOM. Далі браузер поєднує DOM і CSSOM, створюючи Render Tree — дерево видимих елементів з відповідними стилями.

Після цього браузер починає обробляти JavaScript-код, який є ключовим елементом у клієнтському рендерингу, оскільки відповідає за побудову інтерфейсу та завантаження контенту. JavaScript може мати атрибути async або defer. Скрипти з атрибутом defer завантажуються паралельно з парсингом HTML і виконуються після побудови DOM, тоді як async-скрипти завантажуються та виконуються відразу після завантаження, незалежно від завершення парсингу HTML [7].

Коли JavaScript оновлює DOM, наприклад, додаючи нові елементи чи змінюючи текст, браузер відображає ці зміни. Процес включає кілька кроків:

1. Оновлення Render Tree: браузер оновлює дерево для відображення нових або видалення зайвих елементів.

2. Перерахунок стилів: якщо змінюються CSS-властивості, браузер перераховує стилі для оновлених елементів.

3. Розміщення: визначаються розміри й позиції елементів.

4. Перерисовка: браузер додає кольори, зображення та інші візуальні елементи.

Після початкового рендерингу сторінка стає готовою для взаємодії. JavaScript обробляє події (наприклад, кліки або введення тексту), що дозволяє створювати динамічний контент без перезавантаження сторінки:

5. При обробці подій JavaScript реагує на дії користувача, змінюючи DOM, наприклад, завантажуючи нові дані під час кліку.

6. Динамічне оновлення інтерфейсу: якщо користувач змінює фільтри на сторінці магазину, JavaScript може надіслати запит на сервер для отримання відфільтрованих даних та оновити відображений список без перезавантаження сторінки.

На рис. 2 зображено механізм роботи клієнтського рендерингу в браузері.

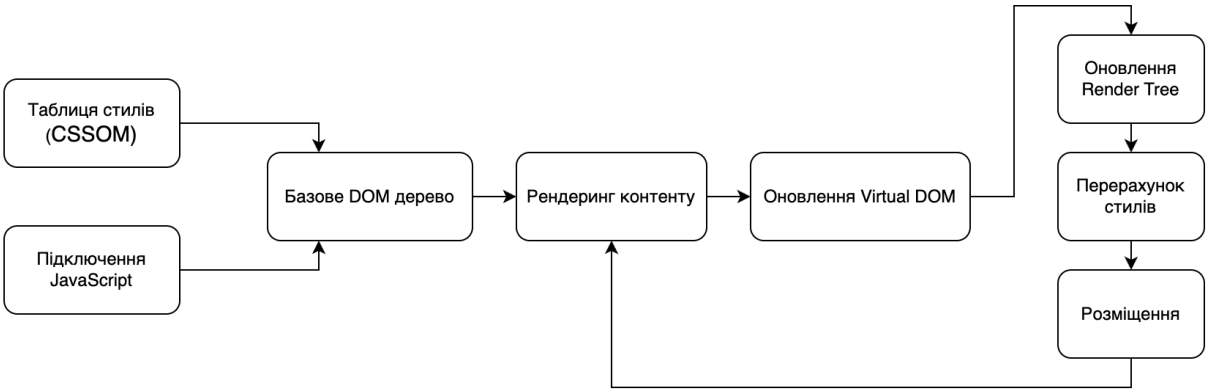

Рис. 2. Механізм роботи клієнтського рендерингу в браузері

Джерело: розроблено авторами.

Проблеми з пошуковими системами. У клієнтському рендерингу проблеми з SEO(Search Engine Optimization) виникають через те, що браузери та пошукові системи індексують вебсторінки. Оскільки клієнтський рендеринг передбачає побудову контенту та інтерфейсу за допомогою JavaScript після завантаження початкового HTML, основний контент стає доступним лише після виконання цього JavaScript-коду [8]. Це створює кілька значних проблем для SEO, оскільки більшість пошукових систем орієнтується на традиційний HTML, щоб швидко та ефективно індексувати сторінки:

1. Відтерміноване завантаження контенту: на відміну від серверного рендерингу, де HTML-документ містить усі важливі дані відразу, при клієнтському рендерингу HTML спочатку завантажується в мінімальному обсязі, а повноцінний контент підтягується та

формується JavaScript'ом. Це означає, що при першому завантаженні сторінка може виглядати "порожньою" або мінімальною, поки JavaScript не виконає завантаження необхідного контенту з API або серверу.

2. Виконання JavaScript ботами пошукових систем: сучасні пошукові системи, такі як Google, здатні виконувати JavaScript, щоб індексувати сторінки з клієнтським рендерингом. Однак цей процес є складнішим і вимагає більше ресурсів, тому обробка JavaScript-контенту часто виконується із затримкою або в обмеженому обсязі. Обмеження виконання JavaScript: Виконання JavaScript вимагає більше ресурсів, ніж парсинг звичайного HTML, тому Google може обмежувати частоту та глибину обробки JavaScript на певних сайтах. Друга хвиля індексації: Часто сторінки з JavaScript проходять через "другу хвилю індексації", коли Google спочатку індексує базовий HTML, а пізніше повертається для обробки JavaScript. Це може створювати затримки в індексації контенту і, як результат, уповільнювати відображення сторінки в результатах пошуку.

3. Проблеми з метатегами та попередніми переглядами сторінок: метатеги які критично важливі для SEO, також зазвичай генеруються за допомогою JavaScript у клієнтському рендерингу. Оскільки JavaScript не виконується відразу, пошукові системи можуть не побачити ці метатеги вчасно для першочергової індексації. Це може негативно вплинути на ранжування сторінки та її привабливість у результатах пошуку, адже користувачі бачать саме метаопис у сніпеті.

4. Зменшена доступність контенту для зовнішніх сервісів: контент, що створюється динамічно через JavaScript, часто стає доступним для індексації та перевірки тільки після завантаження та виконання коду. Це ускладнює доступ до такого контенту для сторонніх сервісів, що також може включати сторонні системи аналітики, сервісів попереднього завантаження або систем, які використовуються для аудиту SEO. Зокрема, ключові елементи SEO і внутрішні посилання, що генеруються JavaScript, можуть бути не доступні для індексації під час первинного сканування, що знижує релевантність сторінки та може погіршити її позицію в результатах пошуку.

5. Вплив на швидкість сторінки та користувацький досвід: швидкість завантаження сторінки є важливим фактором для SEO, а клієнтський рендеринг часто призводить до затримок у відображенні контенту. Це може спричинити високий показник відмов, оскільки користувачі не бажають чекати, поки сторінка повністю завантажиться і відобразить основний контент. Google враховує ці поведінкові метрики у своїх алгоритмах ранжування, тому клієнтський рендеринг, який призводить до повільного завантаження сторінки, може мати негативний вплив на SEO.

Можливі рішення для покращення SEO при клієнтському рендерингу - це серверний, гібридний та динамічний рендеринг. Ці рішення дозволяють згенерувати контент або частину контенту на сервері та допомагають додатку визначити чи є користувач ботом пошукової системи, і надає йому серверно-рендерену версію сторінки для індексації.

Серверний рендеринг пропонує готовий HTML для ботів – він створює повністю сформовану HTML-сторінку на сервері, яка відправляється браузеру чи пошуковому боту. Це спрощує індексацію, оскільки пошукові системи не потребують виконання JavaScrip. Також при використанні серверного рендерингу контент залишається однаковим для користувачів і пошукових ботів, що запобігає помилкам індексації.

У свою чергу, гібридний рендеринг комбінує характеристики серверного та клієнтського рендерингу, він дозволяє вибірково застосовувати SSR для сторінок, критичних для SEO (наприклад, цільові сторінки, статті), і CSR для менш важливих (особисті кабінети, динамічні компоненти), а також надає легкий доступ до метаданих. Це дає змогу оптимізувати індексацію найважливішого контенту.

Динамічний рендеринг забезпечує не меншу оптимізацію SEO, ніж наведені вище варіанти. Це підхід до рендерингу орієнтований на ботів, які аналізують та індексують сторінки в всесвітній мережі. Сервер генерує HTML лише для пошукових систем і соціальних медіаботів, що забезпечує ідеальні умови для індексації навіть для JavaScript-інтенсивних сайтів. Боти отримують попередньо сформований HTML, а користувачі — клієнтський рендеринг. Це дозволяє зберегти інтерфейс для користувачів і SEO-дружність для пошукових систем.

Найбільший приріст продуктивності SEO забезпечую саме серверний рендеринг, проте гібридний та динаміний рендеринг також вирішують проблему поганої SEO оптимізації, яка притаманна клієнтському рендерингу.

Проблема з безпекою додатка. Додатки з клієнтським рендерингом мають низку потенційних проблем із безпекою, які менш виражені в серверному рендерингу. Нижче будуть наведені основні вразливості додатка з клієнтським рендерингом та порівняння цих проблем із використанням серверного рендерингу.

Оскільки в клієнтському рендерингу основна логіка виконується на стороні клієнта, це створює більшу поверхню для атак. Зловмисники можуть переглядати JavaScript-код, аналізувати його на предмет вразливостей і використовувати ці знання для зловживань, наприклад, викрадення даних або несанкціонованого доступу до функцій програми. У серверному рендерингу більшість бізнес-логіки залишається на сервері, тому клієнт отримує лише готовий HTML, що ускладнює атаки.

Однією з головних проблем клієнтського рендерингу є ризик атак типу XSS, оскільки дані, отримані через API, часто динамічно додаються до DOM. Якщо ці дані недостатньо оброблені, зловмисники можуть вставляти й виконувати шкідливий код. У SSR сервер виконує рендеринг сторінок, попередньо обробляючи й екрануючи дані, що знижує ризик XSS-атак. Крім того, у клієнтському рендерингу конфіденційні дані, такі як токени автентифікації, часто зберігаються на клієнті у локальному сховищі чи куках. Це створює ризик витоку таких даних у разі атак типу XSS або доступу до сховища через інші вразливості. У SSR обробка конфіденційних даних відбувається на сервері, що знижує ризик їхнього витоку.

Ще однією проблемою клієнтського рендерингу є доступність внутрішніх маршрутів. Оскільки маршрути зберігаються на клієнті, зловмисники можуть переглянути їх і спробувати отримати доступ до захищених сторінок, якщо серверний контроль доступу налаштований неправильно. У серверному рендерингу маршрути обробляються сервером, що гарантує перевірку прав доступу перед наданням контенту.

Клієнтський рендеринг також піддається ризикам, пов'язаним із прямими викликами API, оскільки браузер виконує ці запити. Якщо API недостатньо захищений, зловмисники можуть його використовувати для несанкціонованого доступу або виконання небажаних дій. У SSR сервер виступає посередником між клієнтом і API, що додає рівень захисту.

Проблема з використанням додатків з клієнтським рендерингом на слабких пристроях (на прикладі мобільних пристроїв). Використання додатка з клієнтським рендерингом на мобільних пристроях має свої особливості та викликає певні проблеми, пов'язані з обмеженнями ресурсів, продуктивності та досвіду користувача. Основні аспекти цих проблем включають:

1. Обмежена продуктивність мобільних пристроїв: Мобільні пристрої, особливо бюджетні або застарілі моделі, мають меншу обчислювальну потужність у порівнянні з десктопами. Клієнтський рендеринг потребує значної кількості ресурсів процесора та оперативної пам’яті, адже пристрій повинен виконувати багато JavaScript-коду, рендерити компоненти, обробляти анімації тощо [9]. Цей процес може призвести до затримок, підвисань або зниження продуктивності додатка.

2. Енергоспоживання та перегрів: Інтенсивне використання процесора для клієнтського рендерингу вимагає більше енергії, що швидше розряджає батарею пристрою. Крім того, довготривале використання ресурсів може призвести до перегріву мобільного пристрою. Це не лише впливає на комфорт користувача, але й може знижувати загальну продуктивність, адже пристрій автоматично обмежує потужність процесора під час перегріву.

3. Проблеми з мережевим з'єднанням: Клієнтський рендеринг часто передбачає, що значна частина логіки додатка завантажується з сервера. На мобільних пристроях, особливо за умов низької швидкості або нестабільного з’єднання, це може призвести до значних затримок завантаження. Якщо додаток повністю залежить від клієнтського рендерингу, при слабкому з'єднанні він може залишатися неактивним протягом тривалого часу або навіть не завантажитися взагалі.

4. Важкість першого завантаження (Initial Load): У випадку клієнтського рендерингу, браузеру мобільного пристрою потрібно завантажити й обробити весь код додатка перед тим, як користувач побачить початковий вміст. Це створює відчутну затримку під час першого завантаження, особливо для великих додатків. У мобільних мережах це може суттєво збільшити час завантаження і негативно вплинути на користувацький досвід.

5. Обмеження кешування та місця на пристрої: Мобільні пристрої мають обмежене місце для кешування даних браузером, і клієнтський рендеринг вимагає завантаження великої кількості ресурсів, які зазвичай кешуються для пришвидшення роботи додатка. Якщо кеш заповнюється або видаляється через інші додатки чи системні обмеження, це може призвести до повторного завантаження великих обсягів даних, що знову ж таки впливає на продуктивність та швидкість додатка.

На пристроях із низькою продуктивністю важливо звести до мінімуму обчислення, необхідні для рендерингу та відображення вебдодатка, щоб зменшити навантаження на процесор. Існують різні підходи для оптимізації клієнтського рендерингу та зменшення розміру коду на стороні клієнта, але зазвичай навіть після оптимізації залишається великий JavaScript-файл, який потрібно виконувати на пристрої користувача. Найбільш продуктивним рішенням для мобільних пристроїв є використання готових HTML-сторінок, тому підходи серверного рендерингу або статичної генерації сторінок підходять найкраще: вони дозволяють відправляти вже згенерований HTML-файл із мінімальним обсягом JavaScript-коду [10].

Проблема значного обсягу файлів, які передаються мережею для відображення вебсторінки. Це головна проблема клієнтського рендерингу, яка лежить в основі його роботи та побудови проєкту, вона є надзвичайно поширеною, особливо в масштабних додатках [11]. Ці файли можуть швидко зростати в об’ємі через особливості клієнтського рендерингу та способу розробки. Приклади проблем, які призводять до великого обсягу файлів:

1. Повне завантаження бібліотек і компонентів: Часто застосунок містить безліч компонентів і сторонніх бібліотек, які можуть збільшувати розмір фінального JavaScript-файлу. Проєкт додає вагу через обробку стану, контекст і рендеринг компонентів. Якщо не застосовувати розділення коду, все це завантажується разом, навіть коли не потрібне на певній сторінці.

2. Великі залежності: додатки зазвичай покладаються на безліч сторонніх бібліотек для оптимізації розробки. Наприклад, використання бібліотек для навігації, роботи з датами, або бібліотеками утиліт. Якщо бібліотеки підключені без ретельного вибору окремих модулів (tree-shaking), це призводить до завантаження непотрібного коду.

3. Динамічні та анімовані компоненти: Складні анімації, інтерактивні елементи або бібліотеки, які реалізовують цей функціонал, додають значну кількість JavaScript-коду. Інколи для реалізації інтерактивності потрібно імпортувати великі бібліотеки з багатою функціональністю, яку частково не використовують, але вона все одно потрапляє в збірку.

4. Підтримка старих браузерів: Для забезпечення сумісності зі старими браузерами у додатку часто додаються поліфіли. Інструменти, такі як Babel можуть конвертувати сучасний код у більш сумісний із різними браузерами, але це також збільшує загальний обсяг коду.

5. Відсутність оптимізації бандлу: Іноді додаток не проходить достатньо мініфікації, а коментарі, відладочні логи та неефективний код залишаються в файлі. Інструменти, як Webpack або Parcel, можуть автоматично мінімізувати файли і видаляти зайві фрагменти, але якщо вони налаштовані некоректно або не використовуються, це призводить до збільшення розміру.

6. Вбудовані стилі та CSS-in-JS: Використання CSS-in-JS або бібліотек, як styled-components і emotion, збільшує розмір JavaScript-файлів, оскільки стилі обробляються всередині JS. Це дозволяє реалізувати динамічні стилі, але також додає ваги до кожного компонента.

Для вирішення цих проблем існує багато практик, але вони спрямовані скоріше на оптимізацію, ніж на повне вирішення проблеми. Насамперед використовування розділення коду за допомогою вбудованих інструментів у фреймворку, що дозволяє завантажувати компоненти лише за потреби та уникати надмірного коду на початку. Це особливо корисно для завантаження важких компонентів або сторінок, до яких користувач звертається рідко [12]. Також хорошою практикою оптимізації є застосування tree-shaking для видалення невикористовуваного коду з бібліотек – це значно зменшує збірку, якщо, наприклад, використовувати лише окремі функції з бібліотек. Важливо підключати тільки потрібні частини бібліотек, уникаючи імпорту повного пакету, якщо він не потрібен повністю. Базовою потребою оптимізації проєкту є мініфікація та налаштування збирачів, як Webpack чи Parcel, вони дозволяють зменшити об’єм файлу за рахунок видалення пробілів, коментарів і зайвого коду [13]. Використання динамічного імпорту також може пришвидшити завантаження, оскільки код завантажується лише тоді, коли він дійсно потрібен. Для обробки стилів CSS-in-JS варто обмежувати використання, оскільки вони додають JavaScript вагу. Інтернаціоналізацію варто реалізовувати так, щоб завантажувати лише потрібну локалізацію, а не всі мови одразу.

Цілковите вирішення проблем клієнтського рендерингу неможливе, оскільки збірка та рендеринг додатка – це складні процеси, що проходять через кілька етапів та потребують багатьох залежних бібліотек і функцій, які працюють разом для створення кінцевого продукту. Усі ці елементи мають бути об'єднані в єдиний JavaScript файл, який буде надіслано користувачу для побудови додатка. Одним зі способів подолати багато труднощів клієнтського рендерингу є перехід на серверний рендеринг. У цьому підході сторінки вже сформовані на сервері, тому користувач отримує готовий HTML і компактний JavaScript файл, що значно зменшує розмір передаваних даних [14].

Висновки. Кожна з проблем, пов’язаних із використанням клієнтського рендерингу у вебдодатках, має свої особливі причини та впливає на ефективність рендерингу й відображення додатка. Більшість із цих проблем має технічний характер, але також присутні питання логічного та семантичного рівня.

Найбільш серйозною проблемою клієнтського рендерингу є великий розмір JavaScript-файлів, які користувачам потрібно завантажувати для коректного відображення додатка. Це впливає не тільки на якість користувацького досвіду, але й на розвиток інтернет-інфраструктури загалом. Щоденне завантаження великих файлів на мільйонах вебсайтів суттєво збільшує глобальний трафік, що робить клієнтський рендеринг недостатньо оптимальним з погляду прогресу та еволюції вебтехнологій.

Клієнтський рендеринг став справжнім проривом у веброзробці та підходах до створення великих, високонавантажених додатків. Його недоліки не дозволяють йому

стати універсальним рішенням для розробки, проте методи, створені на його основі, вирішили багато з цих проблем і стали одним із найперспективніших напрямів у розвитку вебтехнологій.

Більшість проблем клієнтського рендерингу можна вирішити за допомогою серверного рендерингу та статичної генерації сторінок. Ці методи ускладнюють процес розробки та деплойменту, оскільки вимагають від сервера не тільки забезпечення файлів для побудови сторінки, але й виконання логіки для генерування готових HTML-сторінок. Однак вони значно підвищують продуктивність додатка та сприяють розвитку вебтехнологій.

Список використаних джерел

1. Internet Traffic Statistics: A Look at Data Driving Online Behavior [Electronic resource] // Gitnux.org. – Accessed of mode: https://gitnux.org/internet-traffic-statistics/.

2. Iskandar, T. Comparison between client-side and server-side rendering in the web development / T. Iskandar, M. Lubis, T. Kusumasari, A. Lubis // IOP Conference Series Materials Science and Engineering. – 2020. DOI:10.1088/1757-899X/801/1/012136.

3. Hao Han. Practice and Evaluation of Pagelet-Based Client-Side Rendering Mechanism / Hao Han, Yinxing Xue, Keizo Oyama, Yang Liu // IEICE Transactions on Information and Systems. – 2014. https://doi.org/10.1587/transinf.E97.D.2067.

4. Hanafi, R. Comparison of Web Page Rendering Methods Based on Next. js Framework Using Page Loading Time Test / Roy Hanafi, Abd Haq, Ninik Agustin // Teknika. – 2024. – Vol. 13, № 113(1). – Pp. 102-108. DOI: https://doi.org/10.34148/teknika.v13i1.769.

5. Moore R. Compare and Contrast: CSR, SSR, and SSG in NextJS [Electronic resource] / Rick Moore // Medium. – Accessed of mode: https://medium.com/nerd-for-tech/compare-and-contrast-csr-ssr-and-ssg-in-nextjs-58e3caf2e15e

6. Zou, Y. Virtual DOM coverage for effective testing of dynamic web applications. 2014 International Symposium on Software Testing and / Y. Zou, Z. Chen, Y. Zheng, X. Zhang, Z. Gao // Proceedings of 2014 International Symposium on Software Testing and Analysis, ISSTA. - 2014. – Pp. 60-70. DOI: https://doi.org/10.1145/2610384.2610399.

7. Bhatt, D. ReactJS: A Comprehensive Analysis of its features, Performance, and Suitability for Modern Web Development / D. Bhatt, K. Parekh, M. Minat, B. Patel // Interantional Journal Of Scientific Research In Engineering And Management. – 2023. – Vol. 07. DOI 07.10.55041/ IJSREM25667.

8. Patel, V. Analyzing the Impact of Next.JS on Site Performance and SEO / V. Patel // International Journal of Computer Applications Technology and Research – 2023. – Vol. 12. – Pp. 24-27. DOI:10.7753/IJCATR1210.1004.

9. Ardiyanto, R. Analisa Performasi Metode Client Side Rendering, Server Side Rendering, dan Incremental Static Regeneration dalam Proses / R. Ardiyanto, E. Ardhianto // Website Rendering. Computer Science (CO-SCIENCE). – 2024. – Vol. 4(1). – Pp. 19-27 DOI: 10.31294/coscience.v4i1.2427.

10. Conti, M. Content Delivery Policies in Replicated Web Services: Client-Side vs. Server-Side / M. Conti, E. Gregori, W. Lapenna // Cluster Computing. – 2005. – Vol. 8(1). – Pp. 47-60. DOI: https://doi.org/10.1007/s10586-004-4436-5.

11. Nefe Emadamerho-Atori. What is Clinet-side rendering (CSR)? – Accessed of mode https://prismic.io/blog/client-side-rendering.

12. Vallamsetla K. The Impact of Server-Side Rendering on UI Performance and SEO // Karthik Vallamsetla / International Journal of Scientific Research in Computer Science, Engineering and Information Technology. – 2024. DOI: 10.795-804. 10.32628/CSEIT241051067.

13. Zammetti, F. Modern Full-Stack Development: Using TypeScript, React, Node.js, Webpack, and Docker / F. Zammetti. – 2020. – 396 p. DOI: 10.1007/978-1-4842-5738-8.

14. Venkata Naga Sai Kiran Challa. Comprehensive Analysis of Modern Application Rendering Strategies: Enhancing Web and Mobile User Experiences / Venkata Naga Sai Kiran // Journal of Engineering and Applied Sciences Technology. – 2022. DOI: 10.47363/JEAST/2022(4)248.

References

1. Internet Traffic Statistics: A Look at Data Driving Online Behavior. (n.d.). *gitnux.org.* https://gitnux.org/internet-traffic-statistics.

2. Iskandar, T., Lubis, M., Kusumasari, T., Lubis, A. (2020). Comparison between client-side and server-side rendering in the web development. *IOP Conference Series Materials Science and Engineering, 801.* 012136. 10.1088/1757-899X/801/1/012136.

3. Han, H., Xue, Y., Liu, Y., Oyama, K. (2014). Practice and Evaluation of Pagelet-Based Client-Side Rendering Mechanism. *IEICE TRANSACTIONS on Information and Systems.* E97-D. 2067-2083. 10.1587/transinf.E97.D.2067.

4. Hanafi, Roy & Haq, Abd & Agustin, Ninik. (2024). Comparison of Web Page Rendering Methods Based on Next. js Framework Using Page Loading Time Test, 13, 102-108. 10.34148/teknika.v13i1.769.

5. Moore, R. (2021). Compare and Contrast: CSR, SSR, SSG. *medium.com.* https://medium.com/nerd-for-tech/compare-and-contrast-csr-ssr-and-ssg-in-nextjs-58e3caf2e15e.

6. Zou, Y., Chen, Z., Zheng, Y., Zhang, X., Gao, Z. (2014). Virtual DOM coverage for effective testing of dynamic web applications. *Proceedings of 2014 International Symposium on Software Testing and Analysis, ISSTA 2014.* 10.1145/2610384.2610399.

7. Bhatt, D., Parekh, K., Minat, M., Patel, B. (2023). ReactJS: A Comprehensive Analysis of its features, Performance, and Suitability for Modern Web Development. *Interantional Journal Of Scientific Research In Engineering And Management, 07.* 10.55041/IJSREM25667.

8. Patel, V. (2023). Analyzing the Impact of Next.JS on Site Performance and SEO. *International Journal of Computer Applications Technology and Research, 12,* 24-27. 10.7753/IJCATR1210.1004.

9. Ardiyanto, R., Ardhianto, E. (2024). Analisa Performasi Metode Client Side Rendering, Server Side Rendering, dan Incremental Static Regeneration dalam Proses Website Rendering. *Computer Science (CO-SCIENCE), 4,* 19-27. 10.31294/coscience.v4i1.2427.

10. Conti, M., Gregori, E., Lapenna, W. (2005). Content Delivery Policies in Replicated Web Services: Client-Side vs. Server-Side. *Cluster Computing, 8,* 47-60. https://doi.org/10.1007/s10586-004-4436-5.

11. Emadamerho-Atori, N. What is Client-side Rendering (CSR)? [Electronic resource] / Nefe Emadamerho-Atori // Prismic: Headless Page Builder - Launch and Iterate Faster. https://prismic.io/blog/client-side-rendering

12. Vallamsetla, K. (2024). The Impact of Server-Side Rendering on UI Performance and SEO. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 10,* 795-804. 10.32628/CSEIT241051067.

13. Zammetti, F. (2020). Modern Full-Stack Development: Using TypeScript, React, Node.js, Webpack, and Docker. 10.1007/978-1-4842-5738-8.

14. Challa, V. N. S. K. (2022). Comprehensive Analysis of Modern Application Rendering Strategies: Enhancing Web and Mobile User Experiences. *Journal of Engineering and Applied Sciences Technology, 1-6.* 10.47363/JEAST/2022(4)248.

Отримано 20.12.2024

UDC 004.9

Kyrylo Nasenok[1], Maria Voitsekhovska[2]

[1]PhD student, recipient of the Doctor of Philosophy degree in specialty 122
Chernihiv Polytechnic National University (Chernihiv, Ukraine)
E-mail: kaboo@stu.cn.ua. **ORCID:** https://orcid.org/0009-0004-0972-7086

[2]PhD, Associate Professor of the Department of Information Technologies and Software Engineering
Chernihiv Polytechnic National University (Chernihiv, Ukraine)
E-mail: m.voitsekhovska@stu.cn.ua. **ORCID:** https://orcid.org/0000-0002-1711-101X

CLIENT-SIDE RENDERING ISSUES IN THE MODERN WORLDWIDE NETWORK

Client-side rendering is an approach to rendering web applications, allowing content to be processed and displayed directly in a browser. This method enables web developers to create modular and component-based code that is easily extendable, reusable, and simplifies application maintenance. Client-side rendering has revolutionized the web industry, as evidenced by its widespread adoption: as of 2024, approximately 9.5 million websites, or 8% of all active websites worldwide, use this approach, handling approximately 17% of total global web traffic.

Despite its advantages, client-side rendering has certain limitations. It can affect various aspects of security and SEO optimization due to increased vulnerability to attacks and challenges in search engine indexing. The most significant drawback of this approach is the substantial increase in the size of files required for complete application loading and rendering. While this is not critical for smaller projects, it can be a significant strain on network resources for large, high-traffic sites with millions of daily visitors. This requires careful attention to content optimization and the use of additional tools to maintain stable application performance.

The problem highlights the growth of global web traffic in recent years and the need to optimize this traffic, as it grows faster than the physical communications infrastructure that carries it around the world. It also underscores that while client-side rendering enhances development ease, maintainability, and application performance, it introduces new challenges such as increased application file size, SEO issues, and resource allocation difficulties.

This article provides an overview of current issues with client-side rendering and their impact on the performance and functionality of web applications. It analyses the most common client-side rendering issues, including challenges with search engines, usability on low-performance devices, and the large file sizes required to render and display the application. It also examines practices and approaches for addressing these issues.

Future research should focus on optimising existing solutions and migrating current projects to technologies that address client-side rendering challenges, such as server-side rendering and static page generation. In addition, it is important to investigate potential migration difficulties, as these methods require more server-side processing, which adds additional semantics, configuration and deployment work to the project.

Keywords: *client-side rendering; web application; client application; optimization; worldwide network; security; search engine.*

Fig.: 2. References: 14.