

ПЕРЕВАГИ ВИКОРИСТАННЯ REACT ДЛЯ ОПТИМІЗАЦІЇ РЕНДЕРИНГУ ВЕБІНТЕРФЕЙСУ

Курінна Д.С.

e-mail: daria.kurinna@nure.ua

Харківський національний університет радіоелектроніки, каф. ПІ
м. Харків, Україна

This study aims to evaluate the effectiveness of web interface rendering optimization in modern client-side applications with increasing structural complexity. A comparative analysis of imperative DOM manipulation and the declarative model implemented in React focuses on the Virtual DOM mechanism for minimizing direct modifications of the real DOM. An experimental scenario with a dynamic list of elements was used to measure the number of DOM update operations under both approaches. The results indicate fewer direct DOM manipulations when only a fragment of the interface changes, confirming the efficiency of the declarative rendering paradigm in scalable web systems.

Сучасні вебзастосунки характеризуються високим рівнем інтерактивності та значним обсягом динамічних даних. У таких системах рендеринг інтерфейсу виконується багаторазово внаслідок змін стану компонентів, а кожна модифікація реального DOM супроводжується перерахунком стилів, макету сторінки та повторним відображенням, що створює додаткове навантаження на браузер [1].

У традиційному імперативному підході розробник безпосередньо керує оновленням DOM-структури, створюючи або змінюючи вузли за допомогою JavaScript [2]. Під час роботи з великими списками це може призводити до переформування значної частини DOM та збільшення навантаження на механізм рендерингу браузера.

React реалізує декларативну модель побудови інтерфейсу, відповідно до якої розробник описує бажаний стан UI, а система визначає, які зміни необхідно застосувати до DOM. Ключовим елементом цього підходу є Virtual DOM – віртуальне представлення структури інтерфейсу у вигляді JavaScript-об'єктів, що зберігаються в пам'яті. Після зміни стану компонента формується нове дерево Virtual DOM, яке порівнюється з попередньою версією за допомогою алгоритму diffing [3]. У результаті визначається мінімальний набір відмінностей, що синхронізуються з реальним DOM [4].

Гіпотеза дослідження полягає в тому, що використання механізму Virtual DOM у React зменшує кількість операцій оновлення реального DOM порівняно з прямими імперативними маніпуляціями, що сприяє підвищенню продуктивності вебзастосунку.

Для перевірки гіпотези проведено експериментальне порівняння кількості операцій оновлення DOM в імперативному підході та React на прикладі динамічного списку з 1000 елементів, у якому змінюється лише один елемент. В імперативному варіанті оновлення реалізується шляхом очищення контейнера та повторного створення всіх вузлів списку:

```
const list = document.getElementById("list");

function updateList(items) {
  list.innerHTML = "";
  items.forEach(item => {
    const li = document.createElement("li");
    li.textContent = item;
    list.appendChild(li);
  });
}
```

У такому випадку при зміні одного елемента відбувається повторне створення всіх 1000 вузлів `<li>`, що означає виконання приблизно 1000 операцій оновлення DOM.

Наведений імперативний приклад демонструє один із найгірших сценаріїв оновлення списку. У практичній розробці можливі більш оптимізовані підходи, проте у великому legacy-кодi досить часто зустрічається повне переформування DOM-структури.

Натомість у React реалізація списку має вигляд:

```
function List({ items }) {
  return (
    <ul>
      {items.map(item => (
        <li key={item.id}>{item.value}</li>
      ))}
    </ul>
  );
}
```

Завдяки використанню ключів (key) та алгоритму порівняння Virtual DOM при зміні одного елемента оновлюється лише відповідний вузол. Тому кількість модифікацій реального DOM не залежить від розміру списку. Алгоритм diffing у найгіршому випадку має складність $O(n)$, однак використання евристик і ключів дозволяє зменшити обсяг перевірок та витрати на reconciliation. Результати порівняння наведено в табл. 1.

Таблиця 1 – Порівняння кількості операцій оновлення DOM

Кількість елементів	Імперативний підхід	React (Virtual DOM)
1000	≈ 1000 операцій	≈ 1 безпосередня модифікація DOM
5000	≈ 5000 операцій	≈ 1 безпосередня модифікація DOM
10000	≈ 10000 операцій	≈ 1 безпосередня модифікація DOM

Як видно з табл. 1, зі збільшенням кількості елементів інтерфейсу витрати на повне переформування DOM зростають пропорційно розміру списку. У термінах алгоритмічної складності такий підхід характеризується лінійною залежністю $O(n)$, де n – кількість елементів інтерфейсу. Натомість при використанні механізму Virtual DOM оновлення інтерфейсу відбувається шляхом синхронізації лише тих вузлів, які фактично зазнали змін. У разі модифікації одного елемента це призводить до виконання лише однієї безпосередньої модифікації реального DOM, що зменшує обсяг операцій рендерингу.

Водночас механізм Virtual DOM потребує додаткових обчислювальних ресурсів для зберігання віртуального дерева та виконання алгоритму порівняння [5]. У деяких випадках оптимізований імперативний код без використання бібліотек може демонструвати порівнянню або навіть вищу продуктивність. Таким чином, переваги React проявляються насамперед у складних застосунках із частими оновленнями інтерфейсу.

Отримані результати підтверджують ефективність використання механізму Virtual DOM для оптимізації рендерингу інтерфейсу. Порівняльний аналіз показав зменшення кількості операцій оновлення реального DOM при використанні React порівняно з імперативним підходом. Перевага React полягає не у прискоренні роботи браузера, а в оптимізації синхронізації стану інтерфейсу з даними та мінімізації витратних операцій рендерингу.

Практична значущість результатів полягає у підтвердженні доцільності використання декларативної моделі під час проєктування масштабованих клієнтських застосунків із високими вимогами до продуктивності та підтримованості коду.

Список використаних джерел:

1. Document Object Model (DOM). Mozilla Developer Network. URL: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model (дата звернення: 03.03.2026).
2. Flanagan D. JavaScript: The Definitive Guide. 7th ed. Sebastopol: O'Reilly Media, 2020. 704 p.
3. Reconciliation. React Legacy Documentation. URL: <https://legacy.reactjs.org/docs/reconciliation.html> (дата звернення: 03.03.2026).
4. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. 2nd ed. Sebastopol : O'Reilly Media, 2020. 310 p.
5. Kumar T. Fluent React: Build Fast, Performant, and Intuitive Web Applications. Birmingham : Packt Publishing, 2024. 456 p.