DOI: https://doi.org/10.26642/ten-2025-1(95)-259-265
УДК 004.51

М.В. Гаватюк, магістрант
І.О. Саяпіна, к.т.н., доц.
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Удосконалений метод цільового оновлення інтерфейсу користувача для підвищення ефективності вебзастосунків на основі реактивних потоків та віртуального DOM

На даний момент у світі існує безліч вебзастосунків, які в основному зосереджені на взаємодії даних та користувача. Саме через це гостро постає питання маніпуляції даних та їх швидкості оновлення на сторінці користувачів. Тому стаття присвячена розробці удосконаленого та ефективного методу оптимізації оновлення графічного інтерфейсу користувача шляхом поєднання реактивного програмування, віртуального DOM (Document Object Model) та власної системи керування станом. Проаналізовано сучасні підходи до оновлення інтерфейсу, зокрема пряме маніпулювання реальним DOM як засіб оновлення інтрерфейсу найпростішим способом та типові стейт-менеджери, що дають змогу зберігати дані вебзастосунків структуровано та в одному місці. Запропоновано модифікований метод, що дозволяє вибірково оновлювати лише змінені компоненти інтерфейсу, зменшуючи навантаження на систему. Проведено порівняльне дослідження ефективності нового підходу щодо існуючих рішень. Надано також можливості його подальшого вдосконалення та інтеграції у сучасні вебтехнології.

Ключові слова: *оптимізація інтерфейсу; віртуальний DOM; реактивне програмування; швидкість рендерингу; система управління станом; веброзробка*.

Актуальність теми. У сучасному світі цифрових технологій ключовим завданням є створення ефективних та інтерактивних графічних інтерфейсів користувача. У часи, коли користувачі вимагають миттєвого реагування та безперервної взаємодії, підвищення продуктивності інтерфейсу стає основною вимогою до програмного забезпечення. Водночас складність таких інтерфейсів створює труднощі при оптимізації процесів оновлення, особливо під час роботи з великими обсягами даних.

Особливо актуально це для динамічних інтерфейсів, де зміни відбуваються часто й локально, а затримки в оновленні знижують якість взаємодії. Застосування методів, які повністю перерендерюють елементи або напряму змінюють DOM (Document Object Model) при кожній зміні даних, призводить до надлишкових обчислень та зайвого навантаження на браузер. У процесі розробки це ускладнює побудову модульної структури, підвищує кількість ручної оптимізації та знижує гнучкість повторного використання компонентів. Під час тестування виникають труднощі з ізольованою перевіркою логіки відображення, адже кожен стан UI (User Interface) часто залежить від множини сторонніх змін. Масштабування таких рішень стає проблематичним. Зі збільшенням кількості елементів зростає час реакції інтерфейсу, що погіршує користувацький досвід. У довготривалій перспективі підтримка подібних систем ускладнюється через щільну взаємозалежність частин інтерфейсу, що вимагає постійного контролю за наслідками кожного оновлення.

Існуючі бібліотеки та фреймворки, такі як React, Vue, Angular, хоча й пропонують абстракції типу віртуального DOM і стейт-менеджерів, часто потребують ручного налаштування оптимізації, що робить їх непридатними для легких інтеграцій у сторонні проєкти або частково динамічні системи. Крім того, більшість рішень орієнтована на повну заміну архітектури або вимагає використання власних DSL або шаблонів.

На практиці це означає, що не існує універсального, простого для інтеграції методу, який би дозволяв цільове оновлення інтерфейсу без прив’язки до фреймворку, з мінімальними ресурсними витратами та максимальною контрольованістю оновлень. Більшість рішень залишається або надто абстрактними, або надто специфічними.

Запропонований у роботі метод поєднує переваги реактивного управління станом, віртуального DOM і механізму вибіркового оновлення та використовує алгоритми як вдосконалення існуючих рішень. Він дозволяє уникати зайвих рендерів, забезпечує гнучку інтеграцію у вже існуючі системи та покращує керованість змін.

Цільовою сферою застосування методу є вебінтерфейси, що потребують частих, але локальних оновлень: адміністративні панелі, системи моніторингу, дашборди, таск-менеджери та інші подібні системи. Саме для таких сценаріїв важливою є висока швидкодія, чітка логіка оновлення й легкість обслуговування коду.

Таким чином, тема є актуальною не лише в контексті технологічного розвитку вебінтерфейсів, а й у площині практичного вдосконалення процесів розробки, тестування та підтримки сучасного динамічного програмного забезпечення.

Аналіз останніх досліджень та публікацій, на які спираються автори. Упродовж останнього десятиліття питання продуктивності користувацького інтерфейсу стало ключовим у розробці frontend-додатків, про що свідчить дана стаття [1].

У сучасній літературі багато хто розглядає певну структуру оптимізації вебзастосунків, яка охоплює різні етапи розробки та впровадження [2, 3]. Зокрема вирізняють сучасні інструменти, методики та підходи до оптимізації продуктивності вебінтерфейсів [4], враховуючи використання віртуального DOM, управління станом та реактивне програмування [5].

Найпоширенішим підходом до вирішення проблеми надмірного оновлення стала концепція віртуального DOM [6], що набула широкого застосування завдяки бібліотеці React [7]. Пізніше цю ідею реалізували й інші фреймворки, а саме: Vue та Inferno.

Окрему нішу займають фреймворки, що активно впроваджують реактивність на рівні ядра [8]. Прикладом є Svelte [9], Solid.js [10] та сучасний Angular із Signals API. Вони дозволяють відстежувати зміни стану з високою точністю, мінімізуючи кількість оновлень у DOM. Проте навіть ці підходи часто базуються на загальних рішеннях, що не завжди оптимально підходять для специфічних проєктів із великою кількістю взаємодій та складною логікою стану.

У відповідь на ці виклики з'являються дослідження, присвячені створенню кастомних state-менеджерів [11] та легких віртуальних дерев. Основна ідея – мати повний контроль над потоком оновлень, зберігаючи при цьому простоту та передбачуваність у коді. Такі рішення рідко стають популярними, однак демонструють високий потенціал у проєктах, де продуктивність має критичне значення [12].

Метою статті є розробка та обґрунтування удосконаленого методу цільового оновлення графічного інтерфейсу користувача, який забезпечує високу продуктивність, точність візуальних змін і простоту інтеграції без прив'язки до конкретного фреймворку. Запропонований підхід базується на поєднанні реактивної моделі передачі стану, віртуального DOM і легковагого механізму вибіркового рендерингу. Його впровадження спрямоване на усунення надмірних оновлень, зменшення навантаження на DOM, покращення масштабованості рішень і забезпечення контрольованості оновлень у складних та динамічних інтерфейсах.

Для досягнення цієї мети необхідно виконати такі завдання: дослідити наявні методи оптимізації інтерфейсів, зокрема ті, що використовують управління станами, віртуальний DOM та реактивне програмування; розробити підхід для інтеграції цих методів та їх вдосконалення з метою забезпечення максимальної ефективності інтерфейсу; провести експериментальну оцінку ефективності запропонованих рішень.

Викладення основного матеріалу. У межах цього дослідження пропонується вдосконалений метод оптимізації оновлення графічного інтерфейсу користувача, який поєднує механізми віртуального DOM, реактивного управління станом і вибіркового рендерингу компонентів. Метод спрямований на усунення типових проблем продуктивності, пов'язаних із надмірним перерендеренням та дублюванням логіки оновлення, що виникають при традиційних підходах у побудові інтерфейсів. Основною перевагою є здатність точно визначати компоненти, що потребують оновлення, і оновлювати виключно їх, залишаючи незмінні частини інтерфейсу недоторканими.

Ключова ідея полягає у створенні спрощеного та легковагого інструменту, який дозволяє точково синхронізувати інтерфейс із поточним станом застосунку без потреби в надлишкових обчисленнях. Реалізація цього підходу базується на трьох складових: абстрактному представленні інтерфейсу у вигляді віртуального дерева (vDOM), реактивній моделі передачі змін через потоки даних і кастомному централізованому сховищі стану.

Метод розроблено як відповідь на обмеження сучасних бібліотек, таких як React, Vue чи Redux, які хоча й дозволяють будувати ефективні графічні системи, однак у складних випадках вимагають ручної оптимізації та глибокого контролю з боку розробника. У нашій реалізації було втілено автоматизоване виявлення змін у стані, їх точкову маршрутизацію до відповідних компонентів і оновлення інтерфейсу через оптимізовану систему дифінгу.

На відміну від типових підходів, де зміна стану часто провокує повне оновлення дерева елементів або вимагає складних механізмів порівняння, розроблений метод поєднує централізоване реактивне сховище, що побудоване на потоках даних, з віртуальним поданням інтерфейсу (Virtual DOM), доповненим алгоритмом ефективного виявлення змін. Ключовою інновацією є впровадження системи цільового рендерингу, в якій оновлення реального DOM здійснюється лише для тих елементів, що були змінені, з використанням попередньо визначених ключів і алгоритму пошуку найдовшої спільної підпослідовності (LCS).

Принцип роботи запропонованого методу можна умовно поділити на кілька етапів. Перш за все усі компоненти інтерфейсу описуються у вигляді віртуального дерева, що містить мінімально необхідну інформацію про кожен елемент: тег, властивості, унікальний ключ та вкладеність. Це дерево не має прямого зв'язку з DOM, що дозволяє проводити з ним численні операції без навантаження на браузер.

```
class VNode {
  constructor(tag, props = {}, children = []) {
    this.tag = tag;
    this.props = props;
    this.children = children;
    this.key = props.key || null;
  }
}
```

Паралельно із цим централізований стан програми зберігається у реактивному сховищі, реалізованому на базі потоків.

```
import { BehaviorSubject } from 'rxjs';

class ReactiveStore {
  constructor(initialState = {}) {
    this._state$ = new BehaviorSubject(initialState);
  }

  getState() {
    return this._state$.getValue();
  }

  selectAll() {
    return this._state$.asObservable();
  }

  select(key) {
    return this._state$.asObservable().pipe(
      map(state => state[key]),
      distinctUntilChanged()
    );
  }

  set(key, value) {
    const currentState = this._state$.getValue();
    this._state$.next({ ...currentState, [key]: value });
  }

  update(newState) {
    this._state$.next(newState);
  }
}
```

Кожна зміна стану ініціює подію, яка поширюється лише до тих компонентів, що на неї підписані. Завдяки цьому досягається вибіркова реактивність: не весь інтерфейс реагує на зміну, а лише відповідні частини. Такий підхід дозволяє значно зменшити кількість оновлень і запобігає ререндерингу невпливових ділянок.

Коли відбувається зміна даних, формується нове віртуальне дерево. Система проводить порівняння старого та нового дерев із використанням ключів та алгоритмів визначення найменшого набору змін (зокрема, модифікованого варіанта алгоритму пошуку найдовшої спільної підпослідовності). Це дозволяє з високою точністю виявити, які саме вузли потрібно оновити у реальному DOM.

```
function lcs(a, b) {
  const m = a.length, n = b.length;
  const dp = Array.from({ length: m + 1 }, () => Array(n + 1).fill(0));

  for (let i = m - 1; i >= 0; i--) {
    for (let j = n - 1; j >= 0; j--) {
      if (a[i].key === b[j].key) {
```

```
        dp[i][j] = 1 + dp[i + 1][j + 1];
       } else {
        dp[i][j] = Math.max(dp[i + 1][j], dp[i][j + 1]);
       }
      }
     }

     return dp;
    }
```

Оновлення здійснюється через механізм цільового рендерингу, що викликає повторне створення лише змінених елементів і вставляє їх у DOM з мінімальною кількістю маніпуляцій. Це не тільки прискорює процес, а й зменшує споживання пам'яті, оскільки не потребує утримання зайвих копій або дублювання структури.

Архітектурно метод реалізовано у вигляді трьох взаємопов'язаних модулів:

1. VirtualDOM Engine – відповідає за побудову віртуального дерева, збереження попереднього стану і виконання дифінгу;

2. Reactive State – керує змінами стану, транслює оновлення через реактивні потоки та забезпечує підписку компонентів;

3. Component Dispatcher – маршрутизує події, відповідає за відтворення змінених частин інтерфейсу у реальному DOM.

Особливу увагу приділено механізму підписки. Кожен компонент реєструється лише на ті фрагменти стану, які йому дійсно потрібні. Це забезпечує низьку зв'язаність системи та високу гнучкість. Наприклад, зміна у піддереві завдань не зачіпає глобальні елементи навігації або фільтрації.

Крім того, реалізовано механізм мікробатчингу, який відтерміновує оновлення до найближчого кадру анімації через requestAnimationFrame. Це дозволяє групувати кілька змін в один оновлювальний цикл, що ще більше знижує навантаження на DOM і забезпечує плавність візуального відгуку.

```
class MicroBatcher {
 constructor() {
  this.queue = [];
  this.scheduled = false;
 }

 schedule(task) {
  this.queue.push(task);
  if (!this.scheduled) {
   this.scheduled = true;
   requestAnimationFrame(() => this.flush());
  }
 }

 flush() {
  this.queue.forEach(fn => fn());
  this.queue = [];
  this.scheduled = false;
 }
}
```

На відміну від Redux, де зміни фіксуються у вигляді суворих редюсерів та глобального дерева дій, у нашому підході використовуються легші механізми без надмірного формалізму. Водночас забезпечується передбачуваність змін і простежуваність – кожна зміна стану має конкретне джерело і реактивно тригерить відповідну частину інтерфейсу.

Ще одним важливим елементом є підтримка ключів. При генерації віртуального дерева кожному компоненту надається унікальний ключ, що дозволяє швидко знаходити відповідність між старою та новою версією дерева при оновленні. Це є основою для реалізації ефективного дифінгу без повного перебудування дерева.

```
function updateChildren(oldChildren, newChildren) {
 const oldMap = new Map();
 oldChildren.forEach(child => {
  if (child.key !== null) {
   oldMap.set(child.key, child);
  }
 });
```

```
  return newChildren.map(newChild => {
    const oldChild = oldMap.get(newChild.key);
    return diff(oldChild, newChild);
  });
}
```

Для всебічного обґрунтування переваг розробленого методу було проведено порівняння з поширеними архітектурними рішеннями, які використовуються у фронтенд-розробці для оновлення інтерфейсу користувача: Redux, MobX та VueX. Кожен із цих підходів має свої сильні сторони, однак також демонструє певні обмеження в аспектах реактивності, гнучкості та точності оновлень.

Redux – це бібліотека керування станом із суворою структурою: єдиний глобальний стор, уніфікований обіг дій, функції-редюсери та іммутабельність даних. Незважаючи на прозорість і передбачуваність змін, Redux не забезпечує вбудованої реактивності. Як наслідок, при кожному оновленні частини стану зміни передаються всім підписаним компонентам, навіть якщо це не має для них значення. Щоб уникнути цього, розробникам доводиться вручну оптимізувати підписки, використовувати мемоізацію та додаткові селектори. У запропонованому методі така надмірність усувається автоматично – компоненти підписані лише на ті дані, які безпосередньо використовують, що забезпечується через реактивні потоки та селективне поширення змін.

MobX застосовує концепцію автоматичного відстеження залежностей на основі observable-структур. Це зручно, проте часто призводить до прихованих зв'язків, які складно контролювати й тестувати. Крім того, у випадку неакуратного моделювання стану, MobX схильний до надлишкових оновлень, оскільки не завжди чітко ізольовує зміни. Розроблений підхід, навпаки, забезпечує явну реактивність – підписки виконуються вручну через RxJS, що підвищує прозорість зв'язків між станом і представленням.

VueX, хоч і є потужним стейт-менеджером, сильно прив'язаний до екосистеми Vue.js. Його підхід також базується на централізованому сховищі та мутаціях стану через визначені канали. Така інтегрованість обмежує повторне використання чи адаптацію цього підходу в проєктах, не пов'язаних з Vue. У нашому методі архітектура залишається незалежною від конкретного фреймворку, що дозволяє вбудовувати її в будь-який застосунок, разом з класичними багатосторінковими сайтами.

Таким чином, порівняно з альтернативами запропонований метод:

1. Усуває надлишкові оновлення, автоматично адресуючи лише потрібні зміни;
2. Не потребує складної конфігурації або обов'язкової інтеграції з фреймворками;
3. Надає гнучкий інтерфейс через RxJS-оператори, які дозволяють реалізовувати складну логіку трансформації потоку даних;
4. Забезпечує явність і контрольованість реакцій інтерфейсу на зміни стану.

У прикладній реалізації метод протестовано на прикладі списку завдань (Todo App), що є типовим кейсом для перевірки ефективності UI-підходів. У межах експерименту було змодельовано сценарії додавання, видалення, сортування та редагування елементів списку на 1000 елементів. Було зафіксовано час оновлення, кількість DOM-операцій та обсяг пам'яті, яку споживає сторінка під час динамічної взаємодії. Результати свідчать про те, що порівняно з традиційним innerHTML-оновленням, а також з Redux-based рендерингом, запропонований метод забезпечує в середньому на 30–40 % меншу кількість змін у DOM, і до 25 % вищу стабільність FPS при масових оновленнях. Це підтверджує ефективність підходу для високонавантажених систем.

Суттєвою перевагою є також простота масштабування. Архітектура не передбачає жорстких залежностей від зовнішніх бібліотек, що робить її придатною для інтеграції в проєкти будь-якого рівня складності – від невеликих односторінкових застосунків до модульних корпоративних систем.

Таким чином, запропонований метод формує цілісну систему оптимізації інтерфейсу, де логіка, стан і візуальне представлення тісно взаємопов'язані, але ізольовані для кращої підтримки та розширення. Він поєднує гнучкість кастомних рішень з ефективністю сучасних архітектурних шаблонів, таких як Flux і Observer pattern.

Висновки та перспективи подальших досліджень. У цьому дослідженні було запропоновано метод підвищення продуктивності оновлення графічного інтерфейсу користувача шляхом інтеграції власної системи управління станом, віртуального DOM та принципів реактивного програмування. Представлений підхід орієнтований на оптимізацію рендерингу, що дозволяє істотно скоротити час оновлення інтерфейсних компонентів та забезпечити більш плавну й надійну взаємодію з користувачем, особливо в умовах динамічної зміни стану елементів. Основна концепція полягала в уникненні зайвих маніпуляцій із реальним DOM, натомість фокусуючись на використанні полегшеного віртуального представлення для цільового оновлення лише тих частин інтерфейсу, які зазнали змін. У перспективі подальші дослідження можуть бути спрямовані на розробку адаптивних алгоритмів управління станом залежно від типу застосунку, глибшу інтеграцію запропонованого рішення у популярні фреймворки (такі як React або Vue), а також вивчення ефективності підходу в умовах розподіленого середовища рендерингу зокрема, у контейнеризованих системах на кшталт Docker або кластерних платформах типу Kubernetes.

Список використаної літератури:

1. *Marang A.Z.* Analysis of web performance optimization and its impact on user experience / *A.Z. Marang* [Electronic resource]. – Access mode : https://kth.diva-portal.org/smash/get/diva2:1228341/FULLTEXT01.pdf.
2. *Шведов І.* Дослідження доцільності, методів та шляхів оптимізації програмного забезпечення задля пришвидшення роботи веб застосунків / *І.Шведов* // Herald of Khmelnytskyi National University. Technical sciences. DOI: 10.31891//2307-5732-2024-337-3-51.
3. *Болюбаш Н.* Методи підвищення продуктивності прогресивного вебзастосунку бібліотеки на основі моделі RAIL / *Н.Болюбаш, М.Олійник* // Information technology and society. – 2023. DOI: 10.32689/maup.it.2023.1.2.
4. *Shivakumar S.K.* Modern Web Performance Optimization / *S.K. Shivakumar*. – Berkeley, CA : Apress, 2020. DOI: 10.1007/978-1-4842-6528-4.
5. *Ollila R.* Modern Web Frameworks: A Comparison of Rendering Performance / *R.Ollila, N.Mäkitalo, T.Mikkonen* // Journal of Web Engineering. – 2022. DOI: 10.13052/jwe1540-9589.21311.
6. ReactJS: A Comprehensive Analysis of its features, Performance, and Suitability for Modern Web Development / IJSREM Journal. DOI: 10.55041/ijsrem2566.
7. React: Performance Optimization [Electronic resource]. – Access mode : https://vaishnavineema.medium.com/react-performance-optimization-4db330ac60b2.
8. *Hochbergs G.* Reactive programming and its effect on performance and the development process / *G.Hochbergs* [Electronic resource]. – Access mode : https://lup.lub.lu.se/luur/download?fileOId=8932147&func=downloadFile&recordOId=8932146.
9. *Białecki G.* Performance analysis of Svelte and Angular applications / *G.Białecki, B.Pańczyk* // Journal of Computer Sciences Institute. – 2021. – Vol. 19. – P. 139–143. DOI: 10.35784/jcsi.2633.
10. *Kryk J.* Multi-aspect comparative analysis of JavaScript programming frameworks – React.js and Solid.js / *J.Kryk, M.Plechawska-Wójcik* // Journal of Computer Sciences Institute. – 2025. – Vol. 34. – P. 68–75. DOI: 10.35784/jcsi.6712.
11. *Docas A.* Managing Global State with Flux and Redux Patterns / *A.Docas, S.Kayode, J.Paul* [Electronic resource]. – Access mode : https://www.researchgate.net/publication/385384161_Managing_Global_State_with_Flux_and_Redux_Patterns.
12. *Chęć D.* The Performance Analysis of Web Applications Based on Virtual DOM and Reactive User Interfaces / *D.Chęć, Z.Nowak* [Electronic resource]. – Access mode : https://link.springer.com/chapter/10.1007/978-3-319-99617-2_8.

References:

1. Marang, A.Z. (2018), «Analysis of web performance optimization and its impact on user experience», [Online], available at: https://kth.diva-portal.org/smash/get/diva2:1228341/FULLTEXT01.pdf
2. Shvedov, I. (2024), «Doslidzhennia dotsilnosti, metodiv ta shliakhiv optimizatsii programnoho zabezpechennia zadlia pryshvydshennia roboty veb zastosunkiv», *Herald of Khmelnytskyi National University. Technical sciences*, doi: 10.31891/2307-5732-2024-337-3-51.
3. Boliubash, N. and Oliinyk, M. (2023), «Metody pidvyshchennia produktyvnosti prohresyvnoho vebzastosunku biblioteky na osnovi modeli RAIL», *Information Technology and Society*, doi: 10.32689/maup.it.2023.1.2.
4. Shivakumar, S.K. (2020), *Modern Web Performance Optimization*, Apress, Berkeley, CA, doi: 10.1007/978-1-4842-6528-4.
5. Ollila, R., Mäkitalo, N. and Mikkonen, T. (2022), «Modern Web Frameworks: A Comparison of Rendering Performance», *Journal of Web Engineering*, doi: 10.13052/jwe1540-9589.21311.
6. ReactJS: A Comprehensive Analysis of its features, Performance, and Suitability for Modern Web Development (2023), *IJSREM Journal*, doi: 10.55041/ijsrem25667.
7. «React: Performance Optimization», [Online], available at: https://vaishnavineema.medium.com/react-performance-optimization-4db330ac60b2
8. Hochbergs, G., «Reactive programming and its effect on performance and the development process», [Online], available at: https://lup.lub.lu.se/luur/download?fileOId=8932147&func=downloadFile&recordOId=8932146
9. Białecki, G. and Pańczyk, B. (2021), «Performance analysis of Svelte and Angular applications», *Journal of Computer Sciences Institute*, Vol. 19, pp. 139–143, doi: 10.35784/jcsi.2633.
10. Kryk, J. and Plechawska-Wójcik, M. (2025), «Multi-aspect comparative analysis of JavaScript programming frameworks – React.js and Solid.js», *Journal of Computer Sciences Institute*, Vol. 34, pp. 68–75, doi: 10.35784/jcsi.6712.
11. Docas, A., Kayode, S. and Paul, J., «Managing Global State with Flux and Redux Patterns», [Online], available at: https://www.researchgate.net/publication/385384161_Managing_Global_State_with_Flux_and_Redux_Patterns
12. Chęć, D. and Nowak, Z., «The Performance Analysis of Web Applications Based on Virtual DOM and Reactive User Interfaces», [Online], available at: https://link.springer.com/chapter/10.1007/978-3-319-99617-2_8

Гаватюк Максим В'ячеславович – магістрант кафедри програмного забезпечення комп'ютерних систем, факультет прикладної математики Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

https://orcid.org/0009-0009-4596-7235.

Наукові інтереси:
- інформаційні системи;
- веборієнтовані системи;
- аналіз даних.

Саяпіна Інна Олександрівна – кандидат технічних наук, доцент, доцент кафедри програмного забезпечення комп'ютерних систем, факультет прикладної математики Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».
https://orcid.org/0000-0003-1541-1681.
Наукові інтереси:
- бази даних;
- інженерія програмного забезпечення.

Havatiuk M.V., Saiapina I.O.

Improved method of targeted user interface updates for enhancing the efficiency of web applications based on reactive streams and virtual DOM

In today's digital landscape, web applications have become an integral part of everyday life, enabling seamless access to information, services, and user interaction. Most modern applications are centered around the dynamic relationship between data and the user interface, which creates significant demands for efficiency in data processing and real-time interface updates. One of the key challenges lies in ensuring rapid, precise, and performance-optimized user interface rendering, especially in systems with frequently changing states. This article focuses on the development and implementation of an efficient update mechanism for graphical user interfaces, aimed at reducing computational overhead while maintaining interactivity and responsiveness. The proposed solution combines three core concepts: reactive programming, which allows automatic propagation of changes through the interface; virtual DOM, which minimizes direct manipulations with the real DOM and accelerates rendering; and a custom state management system designed to handle all application data updates in a centralized and consistent manner. Current practices for DOM manipulation are reviewed, including traditional direct updates and commonly used state managers such as Redux and Vuex. While these approaches are widely adopted, they can introduce unnecessary complexity and overhead in managing application state.

This article introduces a refined method for selective updating only those components of the interface whose state has actually changed. By integrating the strengths of the aforementioned technologies, the method significantly enhances performance, reduces browser load, and improves interface responsiveness even in complex application logic scenarios. Special attention is given to the architectural structure of the solution, the mechanisms for detecting changes, and the evaluation of dependencies between state and UI components.

The article concludes with a comparative analysis of the proposed approach against traditional solutions, demonstrating its effectiveness in real-world usage. Additionally, it outlines the method's scalability, potential for integration into existing front-end frameworks, and prospects for further development in response to growing performance demands in modern web technologies.

Keywords: interface optimization; virtual DOM; reactive programming; rendering performance; state management system; web development.

Стаття надійшла до редакції 02.04.2025.