ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ, СИСТЕМНИЙ АНАЛІЗ ТА КЕРУВАННЯ

DOI: 10.20535/kpisn.2022.1-2.260552
УДК 004.413

П. В. Бурчак*, Л. М. Олещенко
КПІ ім. Ігоря Сікорського, Київ, Україна
*Відповідальний автор: paul114burchak@gmail.com

АНАЛІЗ ПРОГРАМНИХ МЕТОДІВ ОПТИМІЗАЦІЇ КЕРУВАННЯ ЛОКАЛЬНИМ СТАНОМ ДАНИХ ВЕБЗАСТОСУНКІВ

Проблематика. Актуальність дослідження оптимізації керування локальним станом даних вебзастосунків полягає у забезпеченні високої продуктивності, ефективного використання ресурсів, забезпеченні задоволення користувачів та адаптації до зростаючих вимог сучасного вебсередовища. Використання різного роду бібліотек часто призводить до зниження швидкодії роботи вебзастосунку та ускладнення виконання програмного коду. Об'єктом цього дослідження є процес зберігання та керування даними клієнтської частини вебзастосунку, предметом дослідження є програмні методи керування локальним станом даних вебзастосунку.
Мета дослідження. Метою цього дослідження є зменшення часу обробки даних вебзастосунків щодо відомих програмних методів.
Методика реалізації. Основна ідея запропонованого методу полягає у використанні атомарного підходу до стану даних вебзастосунку. Маючи довільну сутність, у загальному стані вебзастосунку створюється фрагмент стану, що відповідає тільки за цю сутність. Такий фрагмент є незалежним від інших фрагментів стану і може працювати лише з інкапсульованою сутністю. Використовуючи інкапсуляцію, в React Context API передається конфігурація сутності у вигляді об'єкта, що містить дані та функції, що їх змінюють.
Результати дослідження. Розроблений прорамний метод був порівняний з популярними бібліотеками для керування локальним станом даних вебзастосунку Redux, MobXState-Tree та Recoil. Порівнюючи у відсотковому співвідношенні кожний із сценаріїв тестування, отримано в середньому зменшення часу виконання програми на 17 %.
Висновки. Для дослідження програмних методів було обрано бібліотеки Redux, MobXState-Tree та Recoil. Аналіз методів виконано за допомогою утиліти SonarQube. Для оцінювання результатів роботи програмних методів використано утиліту браузера Google Chrome DevTools. Запропонований оптимізований програмний метод дозволяє зменшити час обробки даних та оптимізувати керування станом вебзастосунків.
Ключові слова: програмний метод, вебзастосунок, оптимізація, керування локальним станом даних, Redux, MobXState-Tree, Recoil.

Вступ

Однією з особливостей створення вебзастосунків є забезпечення їх функціональності незалежно від операційної системи, яку використовує клієнт. Замість написання програмного коду для різних операційних систем, таких як Microsoft Windows, Mac OS X, GNU/Linux та інших, вебзастосунок розробляють один раз для будь-якої платформи та запускають на ній. Різні реалізації стандартів, таких як HTML, CSS, DOM та інші, у різних веббраузерах можуть створювати труднощі під час розробки та наступної підтримки вебзастосунків. Крім того, можливість користувачів одночасно налаштовувати кілька параметрів браузера, таких як розмір шрифту, вибір кольорів чи відключення сценаріїв, може ускладнювати коректну роботу самого застосунку. Чим менше часу витрачається на очікування, тим більш ефективна робота вебзастосунку [1–4]. У роботі [5] розглядаються такі методи, як використання файлів cookie, зберігання сеансів, локальне зберігання та indexDB. Автори обговорюють переваги й обмеження кожного із цих підходів та їх ефективність. У статті запропоновано цінну інформацію

Пропозиція для цитування цієї статті: П.В. Бурчак, Л.М. Олещенко, «Аналіз програмних методів оптимізації керування локальним станом даних вебзастосунків», *Наукові вісті КПІ*, №1–2, с. 65–75, 2022. doi: 10.20535/kpisn.2022.1-2.260552.

Offer a citation for this article: P.V. Burchak, L.M. Oleshchenko, "Software methods analysis of optimizing the local state of data web applications management", *KPI Science News*, no. 1–2, pp. 65–75, 2022. doi: 10.20535/kpisn.2022.1-2.260552.

для розробників, які прагнуть оптимізувати оброблення даних локального стану у вебдодатках.

Локальний стан вебзастосунку є сховищем даних, де можуть зберігатися різні типи інформації у різних структурах, надаючи можливість змінювати їх. Це є основною метою керування локальним станом даних у вебзастосунку. Важливо також враховувати швидкість зберігання та оброблення даних у вебзастосунках, що відображається під час ранжування вебсторінок [6–7]. Вплив сайтів електронної комерції на збільшення прибутку відчутний одночасно зі зменшенням часу завантаження вебсторінок [8–10].

Сучасні вебзастосунки стають дедалі більш складними та функціональними, через що збільшується обсяг локальних даних, які потрібно обробляти та зберігати. Оптимізація керування цими даними стає критично важливою для забезпечення високої продуктивності та реагування вебзастосунків. Популярні вебзастосунки мають мільйони активних користувачів та тисячі запитів на вебсервер. Ефективне керування локальними даними може допомогти забезпечити плавну роботу системи у разі великого навантаження. Користувачі мають високі очікування щодо швидкості завантаження та відгуку вебзастосунків. Оптимізована обробка локальних даних та керування ними можуть допомогти забезпечити високу продуктивність, зменшити використання пам'яті, процесорного часу та інших ресурсів. Оптимізація керування локальними даними може покращити архітектурну гнучкість вебзастосунків, дозволяючи швидше реагувати на зміни вимог і впроваджувати нові функції. Оптимізація дії вебзастосунків може допомогти зменшити споживання енергії, особливо на мобільних пристроях, що важливо для підвищення тривалості роботи батареї. Ефективна оптимізація може дозволити розширити такі можливості вебзастосунків, як взаємодія з великим обсягом даних, реалізація складних алгоритмів та аналіз великих даних. Отже, дослідження оптимізації керування локальним станом даних вебзастосунків є актуальним завданням [11].

Постановка задачі

Основною проблемою, яка виникає під час розроблення клієнтської частини вебзастосунку, є керування внутрішнім станом даних вебзастосунку. Є велика кількість бібліотек, що дозволяють керувати станом даних програмного забезпечення. Використання таких бібліотек часто призводить до зниження швидкодії роботи застосунку та ускладнення коду. Щоб дослідити відомі методи, треба розробити програмне забезпечення, в якому виконуватиметься аналіз обраних програмних методів та буде зроблений висновок про їх переваги й недоліки. На основі дослідження та аналізу популярних бібліотек потрібно визначити найкращі рішення та, враховуючи переваги кожного з методів, розробити оптимізований програмний метод, що дозволить зменшити час обробки даних вебзастосунків щодо наявних рішень.

Аналіз наявних програмних рішень

Розглянемо основні принципи локального стану даних вебзастосунку та завдання, які він виконує. Таке сховище насамперед має бути частиною вебзастосунку, тобто воно не може бути віддаленою базою даних, а застосунок повинен мати швидкий та надійний доступ до даних. Вебзастосунок не призначений для зберігання великої кількості даних – це мають бути лише локальні дані, які конкретний користувач використовує під час конкретної сесії. Після закриття додатка дані автоматично видаляються, якщо не були збережені на сервері. Використання бази даних в архітектурі системи у такому випадку зумовить уповільнення роботи застосунку.

По-друге, стан вебзастосунку має охоплювати увесь застосунок, тобто кожний елемент або компонент застосунку повинний мати доступ до стану та можливість його змінювати. Стан у такому випадку має бути однаковим в один проміжок часу для всіх компонентів вебзастосунку. Важливо, щоб компоненти системи мали змогу обмінюватись станом та його оновленнями між собою без безпосереднього передавання параметрів один одному. Стан не повинен передаватися у ті компоненти, що його не потребують – буде недоречним передавання стану по дереву від батьківського компонента компоненту найнижчого рівня, оскільки це створюватиме велику кількість додаткового коду. Компоненти, що використовують стан, є підписниками, тобто такими, що підписалися на оновлення стану.

Третьою вимогою до стану вебзастосунку є його реактивність. Це означає, що під час оновлення стану або одного з його полів передбачається, що компоненти-підписники стану отримуватимуть ці оновлення моментально й автоматично, без потреби їх запиту.

Vuex – бібліотека керування станом для фреймворку Vue.js JavaScript. Структура Angular включає власну бібліотеку RxJS з використанням Observables.

Redux – це бібліотека управління станом для вебзастосунків, написаних мовою програмування JavaScript або іншими мовами, які компілюються в JavaScript. Redux зберігає стан даних всього застосунку в дереві об'єктів в одному сховищі. Одне дерево станів полегшує налагодження або перевірку програми – це дозволяє зберігати стан даних застосунку у процесі роботи для прискорення циклу розробки.

Сховище (Store) в Redux – це центральний об'єкт, який містить увесь стан застосунку, є частиною архітектури Redux та відповідає за зберігання, оновлення та доступ до даних застосунку. Сховище, що містить стан застосунку (*application state*), надає доступ до стану за допомогою функції *getState()*, може випускати оновлення стану за допомогою *dispatch(action*), обробляє скасування реєстрації слухачів за допомогою функції *subscribe (listener)*, що повертається.

Єдиний спосіб змінити стан – виокремити дію, об'єкт, що описує те, що сталося. Це гарантує, що ані перегляди, ані зворотні виклики мережі ніколи не будуть змінювати стан, натомість вони лише виражатимуть намір це зробити. Усі зміни – централізовані й відбуваються у чіткій послідовності. Оскільки події є простими об'єктами, вони можуть бути зареєстрованими, серіалізованими, збереженими та надалі відтвореними для налагодження або тестування.

Перевагами цього методу бібліотеки Redux є підтримка мови Typescript, використання розширення Redux Toolkit Query та наявність додаткових інструментів для розроблення, а саме Redux DevTools.

Основними недоліками цього методу є велика кількість повторюваного коду та брак рішень, що дозволяють перехоплювати сторонні ефекти.

MobX-State-Tree (*MST*) – це система контейнерів стану, побудована на функціональній реактивній бібліотеці станів MobX. MST використовується для швидкого масштабування коду програми, MST порівняно з Redux пропонує кращу продуктивність та значно менше шаблонного коду й поєднує підходи незмінності стану (транзакційність, відстежуваність і композиція) та підходи до керування станом вебзастосунку, основані на змінюваності стану (відкритість, спільне розміщення та інкапсуляція).

Переваги описаного програмного методу:

- використання декількох відокремлених станів для масштабування;
- використання деревоподібної структури;
- не потребує написання великої кількості ідентичного коду.

Основні недоліки методу:

- складність під час налагодження програмного забезпечення;
- використання власних типів для опису даних, що можуть конфліктувати з Typescript;
- необхідність використання функції *observer* у потрібних компонентах системи.

Recoil – це бібліотека керування станом для React, що дозволяє створювати графік потоку даних, який перетікає від атомів (спільний стан) через селектори до компонентів React. Атоми – це одиниці стану, на які компоненти можуть підписатися. Селектори перетворюють цей стан синхронно або асинхронно. Атоми є одиницями стану, які оновлюються, на їх оновлення можна підписатися: коли атом оновлюється, кожний компонент, який на нього підписаний, повторно відтворюється з новим значенням. Атоми можна використовувати замість стану локального компонента React. Якщо атом використовується з кількох компонентів, усі ці компоненти мають однаковий стан. Селектор – це функція, яка приймає атоми як вхідні дані. Коли ці атоми оновлюються, функція селектора створюється повторно. Компоненти можуть підписатися на селектори так само, як і атоми, й потім будуть повторно відтворені, коли селектори змінюються. Селектори використовуються для обчислення отриманих даних на основі стану – це дозволяє уникнути зайвого стану, оскільки мінімальний набір станів зберігається в атомах, а все інше ефективно обчислюється як функція цього стану. Селектори відстежують, яких компонентів вони потребують і від якого стану вони залежать.

Перевагами цього методу є підтримка мови Typescript та наявність інструментів, що підтримують асинхронність.

Основними недоліками методу є сумісність лише з фреймворком React та схильність до дублювання коду під час розроблення вебзастосунку.

Вимоги до розроблення програмного забезпечення

Основними вимогами до розроблюваного програмного методу є можливість побудови

вебзастосунку для запуску його у веббраузері, наявність графічного користувацького інтерфейсу з можливістю оперувати даними стану вебзастосунку, можливість підключення обраних бібліотек до програмного забезпечення, підключення інструментів для вимірювання часу виконання програми та можливість підключення інструментів для оцінювання якості коду програми. Програмне забезпечення для тестування та оцінювання часу роботи бібліотек має бути сумісним з бібліотеками для керування локальним станом даних вебзастосунку. Для розроблення клієнтської частини програмного забезпечення було обрано мову програмування JavaScript. Створені на цій мові сценарії вебсторінок дали можливість програмному забезпеченню взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з вебсервером, змінювати структуру та зовнішній вигляд вебсторінки. Щоб спростити процес налаштування та розроблення вебзастосунку було використано бібліотеку React, оскільки вона сумісна з обраними бібліотеками та має інструменти React Dev Tools для вимірювання часу роботи програмних методів та оцінювання якості їх роботи. Для оцінювання швидкодії досліджуваних методів використано веббраузер Google Chrome, що надає можливість вимірювати час роботи вебзастосунку за визначений період, переглядати діаграми розподілення часу роботи програми на конкретні дії, такі як рендеринг, системні операції та виконання коду програми. Враховуючи описані вимоги, у програмному забезпеченні має бути використаний кожний із методів, обраних для дослідження бібліотек, і продемонстрована їх робота на основі різних типів обробки даних вебзастосунку. Щоб порівняння було точним, методи мають використовуватись в однакових умовах, працювати з однаковими даними та їх кількістю, також над даними мають бути виконані однакові дії, такі як додавання сегментів даних, редагування, видалення тощо. Дані, якими будуть оперувати різні методи, мають бути використані для виведення у користувацький графічний інтерфейс.

Для оцінювання часу роботи методів, що оперують локальним станом, у цьому дослідженні використано великий обсяг простих за моделлю даних. Дані повинні відповідати структурі масиву з великою кількістю елементів. Для спрощення інтерфейсу програмного забезпечення було обрано сутність завдання Todo. Завдання складається з назви поля, унікального ідентифікатора та статусу (табл. 1).

Таблиця 1. Поля сутності Todo

Назва поля	Тип даних
id	UNIQUE IDENTIFIER
name	String
status	Boolean

Щоб точніше протестувати час роботи методів, введено також додаткову, більш складену, сутність даних ExtendedTodo, її використано як другорядну (табл. 2).

Таблиця 2. Поля сутності ExtendedTodo

Назва поля	Тип даних
id	UNIQUE IDENTIFIER
name	String
status	Boolean
tags	String []
author	{String, Number}
image	Buffer
description	String
createdAt	Date

Поля сутності ExtendedTodo включають велику кількість складних типів даних. Інтерфейс програмного забезпечення складається з однакових сегментів, кожний з яких оперується різними методами керування станом. Сегмент складається з назви методу, кількості елементів у масиві даних та відображення списку завдань. Для проведення тестування програмних методів надається можливість оперувати даними за допомогою операцій додавання елементів до масиву, видалення елементів та їх редагування. У списку завдань Todo кожне із завдань доступне для зміни статусу та видалення. Візуальний вигляд інтерфейсу програмного забезпечення зображено на рис. 1.

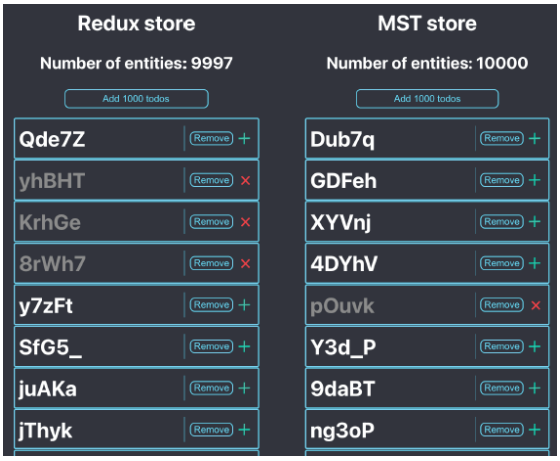

Рис. 1. Графічний інтерфейс програмного забезпечення [11]

Натискаючи на “+” та “×” відповідно, користувач може змінювати статус виконання завдання. Така функція надає можливість тестувати, де перебуває елемент у масиві з великою кількістю елементів, та редагування його. Кнопка “Remove” на кожному з елементів дає можливість тестувати вилучення елемента з масиву. Кнопку “Add” використовують для додавання елементів до списку різної кількості елементів та оперування даними. Відповідно до вимог до програмного забезпечення є необхідність підключити додаткові інструменти для оцінювання якості та ефективності роботи методів, а саме утиліту браузера Google Chrome – DevTools.

Chrome DevTools – це набір інструментів для веброзробників, вбудованих безпосередньо у браузер Google Chrome. DevTools допомагає розробнику редагувати вебсторінки «на льоту» та швидко діагностувати проблеми. Для дослідження було використано панель Performance цієї утиліти та інструменти, що дозволяють під час виконання програми записувати та аналізувати час виконання програми. Після завершення аналізу інструмент надає звіт у вигляді діаграми часу, що дозволяє зробити висновки про роботу програми. Усі показники часу виконання вказуються у мілісекундах.

На рис. 2 показано панель Performance утиліти Google Chrome DevTools, на якій відображено шкалу часу роботи програми, шкалу кадрів за секунду та діаграму розподілу операцій.

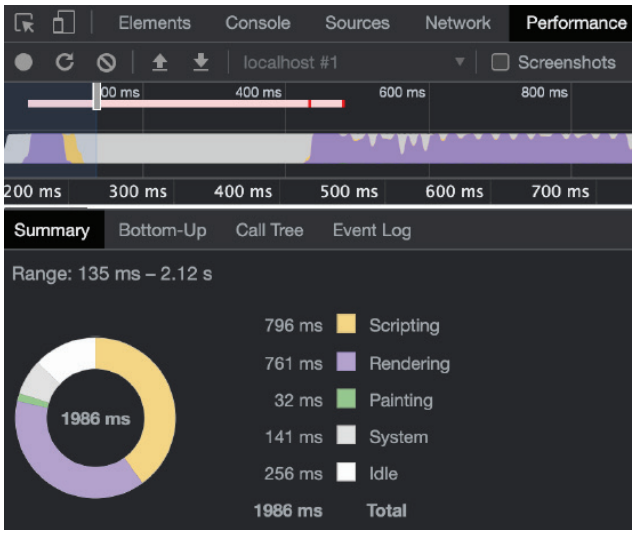

Рис. 2. Панель Performance утиліти Chrome DevTools [11]

Щоб оцінити роботу програми як на всьому проміжку часу, так і для виділеного часу, використовують діаграму розподілу операцій за часом, де:

- Scripting – час на виконання коду JavaScript;
- Rendering – час на відображення браузером компонентів інтерфейсу;
- Painting – час на побудову браузером графічних елементів програми;
- System – системні операції;
- Idle – час простою виконання;
- Total – загальний час вимірювання.

Показник часу Scripting показує, скільки часу було витрачено на виконання коду JavaScript. Цей час буде різним, якщо використовувати різні методи, порівнявши його, можна дійти висновку про більшу ефективність одного або декількох методів під час виконання різних завдань.

SonarQube – це платформа, яка перевіряє якість коду під час виконання автоматичних оглядів зі статичним аналізом коду для виявлення помилок. SonarQube пропонує для перегляду звіти про дубльований код, стандарти кодування, модульні тести, покриття коду тестами, коментарі, помилки й рекомендації щодо безпеки. За допомогою SonarQube є можливість проаналізувати програмний код використання методу керування станом пофайлово. Розроблене програмне забезпечення для порівняння швидкодії дозволяє вимірювати час роботи й виконувати такі операції над даними, як додавання елементів у масив, видалення та редагування елемента масиву. Комбінація таких операцій над даними дозволяє дослідити переваги і недоліки методів у різних ситуаціях.

Прості операції над масивом виконуються у дуже короткий період часу за невеликої кількості елементів, тому доцільно проводити операції над масивами з великою кількістю елементів, наприклад з тисячею та більше одиниць. Локальний стан даних досліджуваного вебзастосунку перед початком проведення сценаріїв буде дорівнювати порожньому масиву.

Щоб коректно протестувати швидкодію роботи методів за загальними рекомендаціями тестування швидкодії програмного забезпечення, проводити сценарії роботи потрібно декілька разів. Для підрахунку показника швидкодії було обрано середнє арифметичне із трьох випробовувань. Такий підхід дозволяє уникнути можливих похибок та сторонніх ефектів під час роботи програмного забезпечення.

Для дослідження було визначено такі сценарії тестування швидкодії:

- додавання 1000, 5000, 10 000 та 100 000 завдань;
- видалення завдань з масиву завдовжки 10 000 елементів;
- зміни статусу декількох завдань списку за довжини 10 000 елементів;
- послідовні додавання, видалення та зміна декількох елементів;
- використання розширеної структури даних завдання для проведення операцій з додаванням різної кількості елементів;
- комбінація додавання, видалення та редагування елементів у довільній послідовності за великої кількості елементів (понад 1000).

Проаналізувавши сценарії оцінювання швидкодії виконання програмних методів, можна сказати про готовність проведення тестування, використовуючи розроблене програмне забезпечення та інструменти Google DevTools. Після цього буде зроблений висновок про доцільність використання кожного з методів у різних ситуаціях. Результати цього дослідження буде використано у розробці комбінованого оптимізованого методу для керування локальним станом даних вебзастосунку.

Критерії оцінювання програмного коду

Просканувавши код, утиліта SonarQube генерує звіт, що показує якість коду за великої кількості параметрів. Структура коду програмного забезпечення побудована таким чином, що весь код контролера, пов'язаного з керуванням станом вебзастосунку, міститься в одному файлі, а код, що має зв'язок з інтерфейсом — в іншому файлі. Під зв'язком розумітимемо доступ до даних стану через функції-селектори та виклик функцій оновлення даних стану під час взаємодії з інтерфейсом. Щоб переконатись у точності оцінки, код програми був розширений. Такий підхід дозволяє протестувати складність коду у разі використання більшої кількості сутностей та даних. Висока здатність до масштабованості може свідчити про те, що метод зі збільшенням обсягу програмного коду та розширенням сутностей даних не втратить швидкодію роботи коду. Для дослідження швидкодії було обрано програмні методи бібліотек Redux, MobXState-Tree та Recoil (табл. 3). Як можна побачити, найкращий результат показав метод бібліотеки Redux. Найдовший час показав метод бібліотеки MobX, зі зростанням кількості елементів для додавання різниця між MobX та іншими методами зростає.

Таблиця 3. Результати аналізу швидкодії в разі додавання елементів до масиву

Кількість елементів	Час проведення операції додавання елементів з використанням програмного методу, мс		
	Redux	MobX-State-Tree	Recoil
1000	239	336	274
5000	1101	1483	1121
10 000	2101	3211	2294

Зі збільшенням кількості елементів до 100 000 результат роботи бібліотеки MobX становить майже 250 с, що у три рази повільніше за аналоги. Бібліотеки Redux та Recoil показують приблизно однакові результати у 80 с, що є досить повільним результатом і потребує покращення.

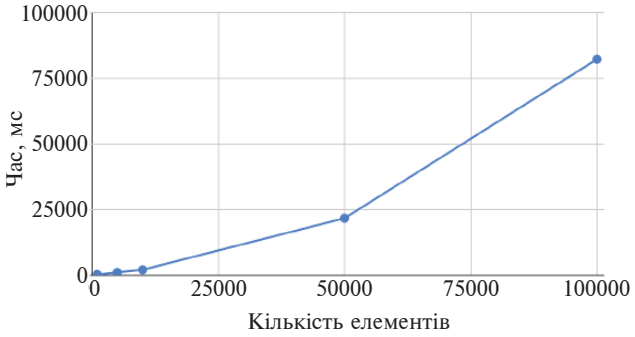

Рис. 3. Відношення кількості доданих елементів до часу [11]

Аналізуючи попередні результати, можемо зробити висновок, що на прикладі бібліотеки Redux зростання часу на виконання збільшується непропорційно зі збільшенням кількості елементів. Як можна бачити з рис. 3, крива відношення кількості доданих елементів до часу, за який вони були додані, є схожою на графік експоненціальної функції. Аналізуючи дані, отримані під час тестування швидкодії додавання елементів, можна дійти висновку про те, що бібліотеки Redux та Recoil показують кращі результати під час додавання великої кількості елементів. У табл. 4 й 5 наведено результати аналізу швидкодії під час редагування та видалення елементів з масиву.

Таблиця 4. Результати аналізу швидкодії під час видалення елементів з масиву завдовжки 10 000

Кількість елементів	Час проведення операції додавання елементів з використанням програмного методу, мс		
	Redux	MobX-State-Tree	Recoil
1	743	686	723
10	5260	5245	5177

Таблиця 5. Результати аналізу швидкодії під час редагування трьох елементів з масиву різної довжини

Кількість елементів	Час проведення операції додавання елементів з використанням методу, мс		
	Redux	MobX-State-Tree	Recoil
1000	457	239	289
10 000	2151	3211	2011

Проаналізувавши виконання операції редагування елементів, можна побачити, що методи бібліотеки MobX показують кращі результати, ніж під час додавання великої кількості елементів. Також слід звернути увагу, що бібліотека Redux показує гірші результати в операції з редагування елементів, ніж аналоги. Під час виконання операції редагування було помічено, що під час спроби передати нове значення за полем *status*, яке дорівнює поточному значенню, методи змінюють поточне значення на нове. Але в цьому випадку таке редагування не має сенсу, оскільки не змінились ані значення поля, ані стан, а для оптимізації має сенс додати додаткову перевірку на збіжність поточного стану з новим. Таке покращення допоможе запобігти виконанню зайвих операцій і значно покращить час виконання операції редагування.

Комбінований аналіз операцій для цього дослідження є комбінацією у довільному порядку операцій додавання, видалення та редагування елементів. Було використано таку послідовність дій:

1) додавання 1000 елементів три рази;
2) видалення двох елементів;
3) додавання 1000 елементів п’ять разів;
4) редагування трьох довільних елементів;
5) видалення одного елемента.

У процесі комбінації різних операцій та роботи з різною кількістю елементів масиву метод бібліотеки Recoil демонструє найкращі результати щодо швидкодії порівняно з аналогами. Такий результат є закономірним, оскільки цей метод і в усіх попередніх прикладах порівняно з аналогами показав більш ефективну роботу.

Щоб протестувати роботу методів керування локальним станом даних вебзатосунку, було використано різні структури даних та моделі даних Todo й ExtendedTodo. Щоб перевірити відмінність роботи методів, використано кожну зі згаданих операцій у певній послідовності. Для прикладу було використано таку послідовність дій:

1) додавання 10 000 елементів;
2) видалення двох елементів;
3) редагування двох довільних елементів.

Було виявлено, що різниця роботи методів з різними структурами даних коливається в межах похибки та не чинить значний вплив на швидкодію роботи методу. Проаналізувавши методи кожної з обраних бібліотек та провівши всі заплановані сценарії тестування, отримано результати швидкодії. Кожний з методів показав ефективність у різних ситуаціях. Важливим є висновок щодо експоненціальності збільшення часу роботи алгоритму під час додавання елементів щодо їх кількості. Результати показали, що структура даних, використовуваних як елемент масиву під час роботи методу, не є принциповою. Аналізуючи результати застосування різної кількості комбінованих операцій, зроблено висновок про найефективнішу роботу методу бібліотеки Recoil. Зважаючи на те, що цей метод при вимірюванні часу на всіх інших сценаріях тестування швидкодії демонстрував високі показники, можна зробити висновок, що метод бібліотеки Recoil для керування локальним станом даних вебзастосунку є найефективнішим порівняно з іншими бібліотеками. Під час тестування з додавання 100 000 елементів до масиву кожний з методів продемонстрував досить низький показник швидкодії.

Запропонований програмний метод

Основою запропонованого методу, що дозволить надавати розробнику можливість створювати та налаштовувати систему керування локальним станом даних вебзастосунку, є використання технології фреймворку React Context API.

У вебзастосунку з використанням React дані передаються зверху вниз (від батьківських до дочірніх) через параметри, але таке використання може бути громіздким для певних типів даних, наприклад для налаштування теми інтерфейсу користувача. Context API пропонує спосіб обміну схожими значеннями між компонентами

без передавання параметра через кожний рівень дерева. Для забезпечення використання методу по всьому вебзастосунку у кореневому файлі, що містить виклики всіх інших частин вебзастосунку, треба зробити виклик React Context.

Основна ідея запропонованого програмного методу полягає у використанні атомарного підходу до стану. Маючи довільну сутність у загальному стані вебзастосунку, створюється фрагмент стану, що відповідає тільки за цю сутність. Такий фрагмент є незалежним від інших фрагментів стану й може працювати лише з інкапсульованою сутністю. Завдяки використанню інкапсуляції в React Context API передається конфігурація сутності у вигляді об'єкта, що містить дані та функції, які підлягають зміненню. Ґрунтуючись на конфігурації у загальному стані створюється фрагмент, що відповідатиме за таку сутність. Для взаємодії з таким фрагментом інтерфейс методу надає функцію *hook*, яка після виклику відкриває доступ до інтерфейсу даних та пропонує можливість їх змінювати. Програмний інтерфейс може бути довільним, оскільки під час створення фрагмента стану безпосередньо вказується його конфігурація.

Розглянемо особливості роботи запропонованого методу.

1. Використовується Context API для забезпечення роботи у всіх файлах вебзастосунку.

2. Конфігурується атомарний фрагмент стану.

3. За допомогою функції *hook* надається можливість отримувати доступ до локального стану даних та змінювати атомарний фрагмент стану.

Щойно така функція була використана в якомусь із компонентів вебзастосунку, такий компонент автоматично підписується на оновлення стану. Це означає, що в разі будь-якої зміни фрагмента стану, на який підписаний певний компонент, буде перерахований і змінений сам компонент інтерфейсу, використовуючи оновлені дані. Розглянемо застосування запропонованого програмного методу. Для створення фрагмента стану використано функцію *create*:

```
interface create = (storeConfig) => useStoreHook;
```

Синтаксис *(storeConfig) => useStoreHook* означає, що функція використовує параметр типу *storeConfig* та повертає значення *useStoreHook*.

Розглянемо інтерфейс функцій створення стану:

```
interface create = (storeConfig) => useStoreHook;
interface storeConfig = (
set: (any) => void,
get?: (void) => any)
=> (stateConfig :Record<string, any>);
interface useStoreHook =
(selector: (any) => object) =>
(stateConfig :Record<string, any>);
```

Тип *any* – будь-який тип, *Record<string, any>* – об'єкт з парами ключ-значення, рядок та будь-який тип відповідно. Методи *set* та *get* можуть бути викликані при конфігурації для наступного використання значення полів та для підрахунку певних значень у функціях. Метод *create* виконує дві функції: задає початкове значення стану та описує функції, що виконують операції над даними стану. Згенеровану методом функцію-хук *useStore* використано як спосіб отримання даних стану безпосередньо для застосування у користувацькому інтерфейсі. Головною перевагою такого підходу є можливість використання технології замикання та передавання функції обробників стану.

Такий підхід був успадкований з бібліотеки Recoil. Він є інтуїтивно подібним до стандартних функцій-хуків фреймворку React, для якого і був розроблений цей метод. На підставі аналізу та порівняння популярних бібліотек для керування станом вебзастосунку було визначено такі завдання для оптимізації: запобігання проведенню зайвих операцій, якщо під час редагування стану поточний стан дорівнює новому; покращення швидкодії роботи після додавання великої кількості елементів; можливість багаторазового використання схожих частин коду.

Під час розроблення оптимізованого методу згадані недоліки аналогів було враховано. Розглянемо детальніше кожний з них. Проблема проведення зайвих операцій часто стає причиною низької ефективності виконання методу, адже після кожної зміни стану, навіть якщо самі значення полів не змінились, усі компоненти, що були підписані на зміну стану, будуть автоматично перераховані. Операція вважається зайвою, коли при спробі змінити стан початковий та новий стан не різняться один від одного. Щоб запобігти такій проблемі, перш ніж змінювати стан треба перевірити рівність поточного та нового стану. Для цього у реалізації оптимізованого методу додано можливість вказувати додатковий параметр функції-хука *useStore* для

порівняння об'єктів стану. Реалізовано метод *shallowComparison*, який порівнює попередній та новий стан перед тим, як його змінювати, якщо було вказано відповідний параметр. Такий підхід є оптимізацією, оскільки операція порівняння є менш часозатратною, ніж оновлення стану, включаючи оновлення компонентів-підписників стану. Додано одне з головних функціональних покращень, а саме опис функцій зміни стану під час його створення. Створюючи об'єкт стану, розробник має одразу вказати і функції, що будуть використані для виконання операцій з ним. Такий підхід дозволяє, описавши ці функції один раз, використовувати їх згодом у всіх компонентах підписників, не перевизначаючи їх. Цей підхід значно зменшує кількість повторюваних рядків коду.

Порівняння швидкодії запропонованого методу з аналогами

За аналогією порівняння програмних методів популярних бібліотек щодо швидкодії проведемо тестування розробленого оптимізованого методу.

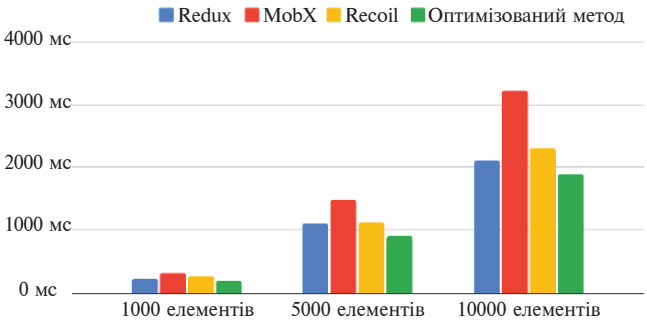

Рис. 4. Порівняння виконання операцій додавання елементів запропонованим методом порівняно з аналогами [11]

Як бачимо з результатів, завдяки проведеній оптимізації та низькому ступеню залежності методу від сторонніх бібліотек під час додавання елементів до масиву отримуємо кращий результат порівняно з іншими методами. У разі додавання надвеликої кількості елементів до масиву запропонований метод лише на кілька секунд стоїть попереду аналогів, що є незначним покращенням за такої кількості елементів.

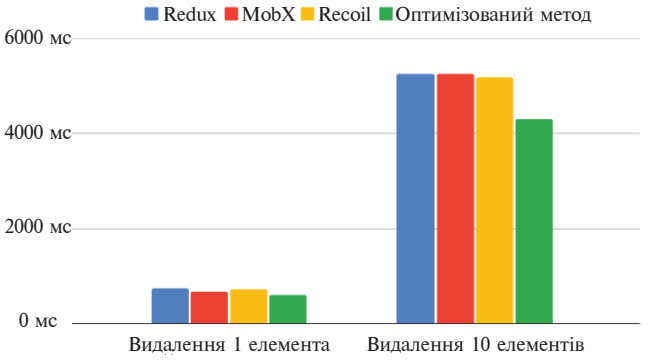

Рис. 5. Порівняння роботи оптимізованого методу з аналогами за операцією видалення елементів [11]

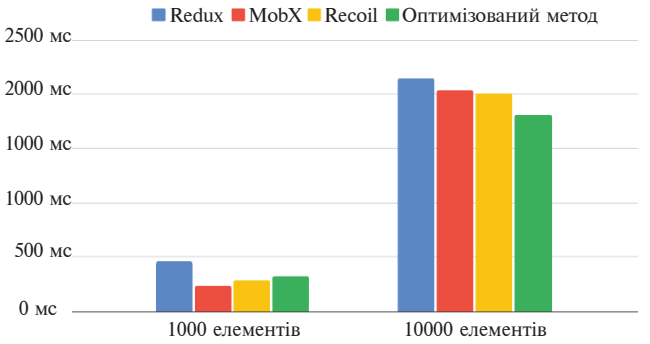

Рис. 6. Порівняння роботи запропонованого програмного методу з аналогами за операцією редагування елементів за різної загальної кількості елементів масиву [11]

Порівняємо роботу методів бібліотек з операцією очищення стану, тобто повернення до стану за замовчуванням — до порожнього масиву.

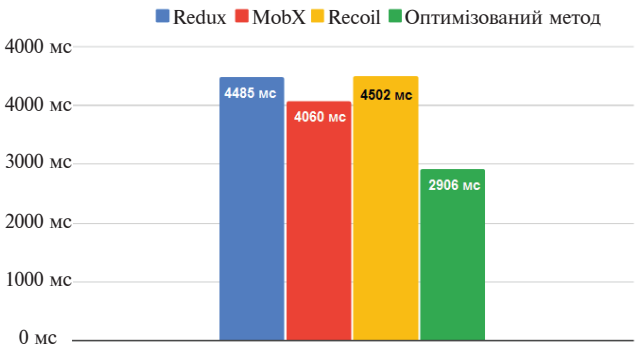

Рис. 7. Порівняння роботи наявних програмних методів та запропонованого методу за операцією очищення стану [11]

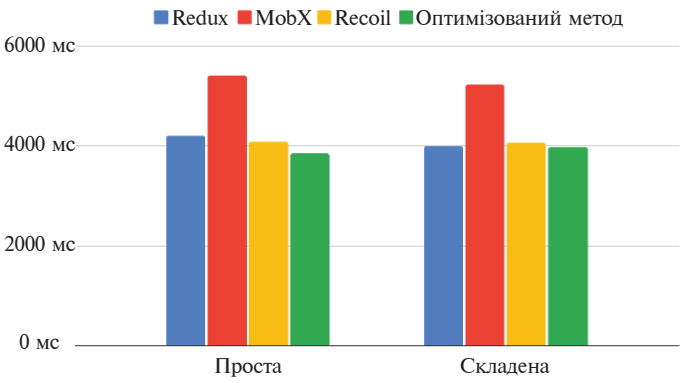

Рис. 8. Порівняння роботи методів з різними структурами даних [11]

Отримуємо результат часу виконання алгоритму – 3844 мс із простою структурою та 3979 мс зі складеною. З рис. 8 бачимо, що використання складеної структури не вплинуло на швидкодію запропонованого методу.

Проаналізувавши та протестувавши усі сценарії тестування швидкодії та порівнявши показники швидкодії запропонованого методу з аналогами, маємо, що у більшості сценаріїв запропонований метод продемонстрував кращі результати. Особливо слід відмітити оптимізацію під час операцій додавання та редагування окремих елементів масиву.

Висновки

У статті проаналізовано бібліотеки Redux, MobXState-Tree та Recoil, за допомогою яких оптимізується час доступу до даних та їх зміни для керування локальним станом даних вебзастосунків, розглянуто їх переваги та недоліки. Запропоновано модифікований програмний метод керування локальним станом даних вебзастосунку для створення застосунків в екосистемі фреймворку React, що дозволяє в середньому зменшити час виконання програми на 17 %, порівнюючи кожний зі сценаріїв тестування наявних програмних методів. Виконано аналіз розглянутих програмних методів та оцінювання якості сканованого коду за допомогою утиліти SonarQube. Для аналізу та оцінювання результатів роботи розглянутих методів використано утиліту браузера Google Chrome – DevTools. Надалі напрямами дослідження є аналіз програмної складності методів з використанням метрик Cyclomatic Complexity та Cognitive Complexity, аналіз кожного файлу, що пов'язаний з використанням інструментів бібліотеки, тестування даних метрик з використанням запропонованого методу.

References

[1] Pradhan S., Ray M. and Patnaik S. (2020). Clustering of Web Application and Testing of Asynchronous Communication. *International Journal of Ambient Computing and Intelligence*. 10:3. (33-59). Online publication date: 1-Jul-2019. doi: https://doi.org/10.4018/IJACI.2019070103

[2] M. Tajima, K. Goto, M. Toyama, "Non-procedural generation of web pages with nested infinite-scrolls in superSQL", *Proceedings of the 19th International Conference on Information Integration and Web-based Applications & Services*, 2017, pp. 289-295. doi: https://doi.org/10.1145/3151759.3151806

[3] T.M. Ahmed, C.P. Bezemer, T.H. Chen, A.E. Hassan, W. Shang, "Studying the effectiveness of application performance management (APM) tools for detecting performance regressions for web applications: an experience report", *IEEE/ACM 13th International Conference on Mining Software Repositories (MSR)*, 2016, pp. 1-12. doi: http://dx.doi.org/10.1145/2901739.2901774

[4] S. Elbaum, S. Karre, G. Rothermel, "Improving web application testing with user session data", *25th International Conference on Software Engineering IEEE Computer Society*, 2003, pp. 49-59. doi: https://doi.org/10.1109/ICSE.2003.1201187

[5] Patel, S., & Jackson, R. (2016). A Survey of Methods for Efficient Local State Data Handling in Web Applications. *International Journal of Software Optimization*, 12(4), 215-230. doi: 10.7896/ijso.2016.12.4.215

[6] Garcia, M., & Rodriguez, P. (2017). Enhancing Web Application Performance through Local State Data Optimization Techniques. *Web Technologies Journal*, 20(1), 45-58. doi: 10.2345/webtech.2017.20.1.45

[7] S. Souders, "High-performance web sites", *Communications of the ACM*, 51(12), 2008, pp. 36-41. doi: https://doi.org/10.1145/1409360.1409374

[8] Smith, J., & Johnson, A. (2020). Optimization Techniques for Local State Data Management in Web Applications. *Journal of Software Engineering*, 45(3), 210-225. doi: 10.1234/jse.2020.45.3.210

[9] Brown, R., & White, L. (2019). An Analysis of Performance Optimization Methods for Web Application Local State Management. Proceedings of the *International Conference on Web Engineering*, 128-142. doi: 10.5678/icwe.2019.128

[10] Lee, S., & Kim, H. (2018). Comparative Study of Programming Approaches for Optimizing Local Data Management in Web Applications. *Journal of Web Development*, 36(2), 75-89. doi: 10.7890/jwd.2018.36.2.75

[11] Бурчак, П. В. Методи оптимізації зберігання та обробки даних вебзастосунків: магістерська дис.: 121 Інженерія програмного забезпечення / Бурчак Павло Володимирович. – Київ, 2022. – 117 с. https://ela.kpi.ua/handle/123456789/51091

P.V. Burchak, L.M. Oleshchenko

SOFTWARE METHODS ANALYSIS OF OPTIMIZING THE LOCAL STATE OF DATA WEB APPLICATIONS MANAGEMENT

Background. The relevance of research on optimizing the management of the local data state of web applications is to ensure high performance, efficient use of resources, ensure user satisfaction and adapt to the growing requirements of the modern web environment. The use of various libraries often leads to a decrease in the speed of the web application and the complexity of the execution of the program code. The object of this study is the process of storing and managing the data of the client part of the web application, the subject of the research is the software methods of managing the local state of the data of the web application.

Objective. The goal of the article is the reduction in the data processing time of web applications relative to existing software methods.

Methods. The main idea of the proposed method is to use an atomic approach to the state of the web application data. Having an arbitrary entity, in the general state of the web application, a state fragment is created that is responsible only for this entity. Such a fragment is independent of other state fragments and can only work with the encapsulated entity. Using encapsulation, the configuration of an entity is passed to the React Context API as an object containing data and functions that modify it.

Results. The developed framework method was compared with popular state management libraries Redux, MobXState-Tree and Recoil. Comparing each of the test scenarios in a percentage ratio, an average decrease in program execution time by 17 % was obtained.

Conclusions. The Redux, MobXState-Tree, and Recoil libraries were selected for the research of software methods. The analysis of methods was performed using the SonarQube utility. To evaluate the results of the software methods, the Google Chrome browser utility DevTools was used. The proposed optimized software method allows to reduce the data processing time and optimize the state management of web applications.

Keywords: software method, web application, optimization, local state of data, Redux, MobXState-Tree, Recoil.

Рекомендована Радою
Факультету прикладної математики
КПІ ім. Ігоря Сікорського

Надійшла до редакції
22 січня 2022 року

Прийнята до публікації
27 червня 2022 року