DOI 10.31891/2307-5732-2024-341-5-31
УДК 004.424

АНАНЧЕНКО ВЛАДИСЛАВ
Міжнародний економіко-гуманітарний університет імені академіка Степана Дем'янчука
e-mail: ananchenko346@gmail.com
ЛОТЮК ЮРІЙ
Міжнародний економіко-гуманітарний університет імені академіка Степана Дем'янчука
https://orcid.org/0000-0001-6696-5583
e-mail: lotyuk@ukr.net

ПЕРЕВАГИ ТА НЕДОЛІКИ REDUX ТА CONTEXT API ДЛЯ УПРАВЛІННЯ СТАНОМ У REACT ЗАСТОСУНКАХ

В роботі наведено результати порівняння бібліотек Redux та Context API, вказані критерії їх порівняння, залежно від специфічних вимог кожного окремого проєкту або окремої частини його функціоналу. Визначені параметри, які слід враховувати при виборі відповідного інструменту, а саме розмір, складність логіки, потреба у масштабуванні, продуктивність, також враховано досвід і компетенції команди розробників.

Ключові слова: Redux, Context API, вимоги до проєкту, функціонал, інструмент розробки, масштабування, продуктивність.

ANANCHENKO VLADYSLAV
International University of Economics and Humanities named after Academician Stepan Demianchuk International University of Economics and Humanities
LOTIUK YURII
International University of Economics and Humanities named after Academician Stepan Demianchuk

ADVANTAGES AND DISADVANTAGES OF REDUX AND CONTEXT API FOR MANAGEMENT AS IN REACT APPLICATIONS

The article examines two prevalent approaches to managing global state in React applications, specifically through the use of Redux and the Context API. It provides a comprehensive comparative analysis of their respective advantages and disadvantages. Managing state, particularly global state, is crucial as it not only affects the overall performance of an application but also impacts development ease and future scalability. Redux, grounded in the principles of Flux architecture, is a well-established tool for global state management. It offers centralized state management via a global store, enabling clear tracking of state changes, avoiding unwanted side effects, and maintaining predictable application behavior, especially in large and complex projects. However, despite its reliability, Redux often leads to an increase in boilerplate code. On the other hand, the Context API, a native React tool, provides a simple and minimalist approach to state management, making it ideal for use in simpler applications or isolated parts of larger projects. It allows for the avoidance of "prop drilling" and facilitates convenient access to context from any point within the component tree. Nonetheless, the Context API may pose performance issues, particularly when numerous components subscribe to a single context, which can lead to inefficiencies as the application scales and new hierarchical levels are added to the component tree. The article explores various scenarios for employing Redux and the Context API in projects of different complexity and scale. Experimental studies demonstrate that in less demanding projects, the Context API ensures swift application performance with minimal configuration. Conversely, in larger projects with complex logic and multi-layered structures, where a single context manages the state of components across different hierarchical levels without partitioning, Redux exhibits superior performance. The article emphasizes the importance of appropriately combining these tools in large–projects. Depending on the project's requirements, a hybrid approach may be effective, such as using the Context API to create independent contexts for managing the state of specific parts of the project or dynamic modules, while employing Redux for managing global state across the entire application. This approach allows for efficient state management throughout the application and achieves an optimal balance between performance and adaptability. The conclusions of the article aim to guide developers in selecting the most suitable tool based on the specifics of their projects. In the diverse landscape of modern web applications and global state management tools, understanding the strengths and limitations of Redux and the Context API is key not only to successfully implementing effective solutions in React applications but also to making informed decisions when choosing third-party state managers, particularly those developed post-Redux.

Keywords: Redux, Context API, project requirements, functionality, development tool, scaling, performance.

Постановка проблеми

У сучасному світі створення веб-застосунків бібліотека React.js все ще займає провідне місце серед інших інструментів, хоча постійна поява нових фреймворків і бібліотек вже стала звичним явищем. React дозволяє розробникам створювати як прості Single Page Applications (SPAs), які будуються виключно за допомогою JavaScript і працюють повністю на стороні клієнта (веб-браузера), так і комплексні, динамічні веб-додатки з різними архітектурними підходами [1, 7]. Зокрема, React віднедавна підтримує створення серверних компонентів за допомогою Server-Side Rendering (SSR), на кшталт того що реалізується, у фреймворку Next.js, побудованому на основі React.js [10, 15]. У цьому контексті React пропонує неабияку гнучкість та можливості для кастомізації у порівнянні з іншими інструментами, такими як фреймворк Angular від Google [11].

Однак зі зростанням складності додатків і їх масштабів, зокрема зі збільшенням кількості компонентів, модулів і їх взаємодій, питання управління станом, особливо глобальним або кількома різними станами, постає особливо гостро. Це вимагає ретельного вивчення та вибору правильного підходу до його управління, так як у сучасних веб-застосунках станів може бути безліч. Вони створюються як за допомогою сторонніх state management, так і з використанням вбудованих можливостей, які пропонує бібліотека React. Крім того, навіть сторонні бібліотеки, що спочатку були розроблені для вирішення своїх окремих завдань, останнім часом також пропонують свої інструменти для цих цілей [21–23].

Управління станом у React-додатках є важливою складовою розробки, оскільки воно визначає, як дані

зберігаються, оновлюються та передаються між компонентами. Стан додатку включає всі значення, що зберігають інформацію про поточний стан інтерфейсу, користувача або системи. Цей стан може бути локальним, тобто обмеженим одним компонентом, або глобальним, який доступний для всього додатку. Також можливі проміжні варіанти, наприклад, використання провайдерів, які розташовуються на певному рівні додатку та огортають лише ті частини і компоненти, які потребують оновлення цього стану і даних з нього [18, 19]. Правильний підхід до вибору інструментів управління станом покращує не лише продуктивність додатку, але й забезпечує зручність написання unit-тестів, а також спрощує його підтримку та розширення.

Існує безліч архітектурних підходів і сторонніх бібліотек для управління станом у React-додатках. Серед них одними з найбільш популярних і широко використовуваних є сторонній Redux та нативний Context API [30–32]. Останніми роками все більше команд відмовляються від використання Redux на користь сучасніших рішень. Проте Redux все ще залишається популярним серед розробників, які вважають його підхід (flow) зручним на етапі розробки та тестування. Незважаючи на те, що цей підхід вимагає написання значної кількості шаблонного коду (boilerplate code), його однонаправленість і передбачуваність приваблюють багатьох фахівців.

Варто зазначити, що обидва підходи – Redux і Context API – мають свої особливості, переваги та недоліки [34, 35]. Це впливає на вибір відповідного інструменту залежно від потреб проєкту. Оскільки ці інструменти створені для різних, хоча й схожих, цілей, вони не можуть вважатися повністю взаємозамінними. З цієї причини їх пряме порівняння може бути не зовсім коректним. Однак, враховуючи, що розробники React при створенні Context API частково мали на меті заміну сторонніх state management рішень у випадках, коли використання останніх є надлишковою справою, доцільно провести аналіз їх відмінностей [25, 26].

Крім того, реалізацію стану в додатку з однаковою поведінкою можна здійснити як з використанням Redux, так і Context API. Це дозволяє провести порівняння цих підходів для визначення оптимального рішення в різних умовах [28, 29].

Redux – це централізована система управління станом, яка базується на концепціях Flux-архітектури (рис. 1). Цей підхід дозволяє зберігати весь стан додатка в одному глобальному сховищі (store), що робить його доступним для будь-якого компонента без необхідності передачі пропсів (props) через усю ієрархію компонентів. Це значно спрощує управління станом у тих місцях, де необхідно керувати складними логічними операціями [33].

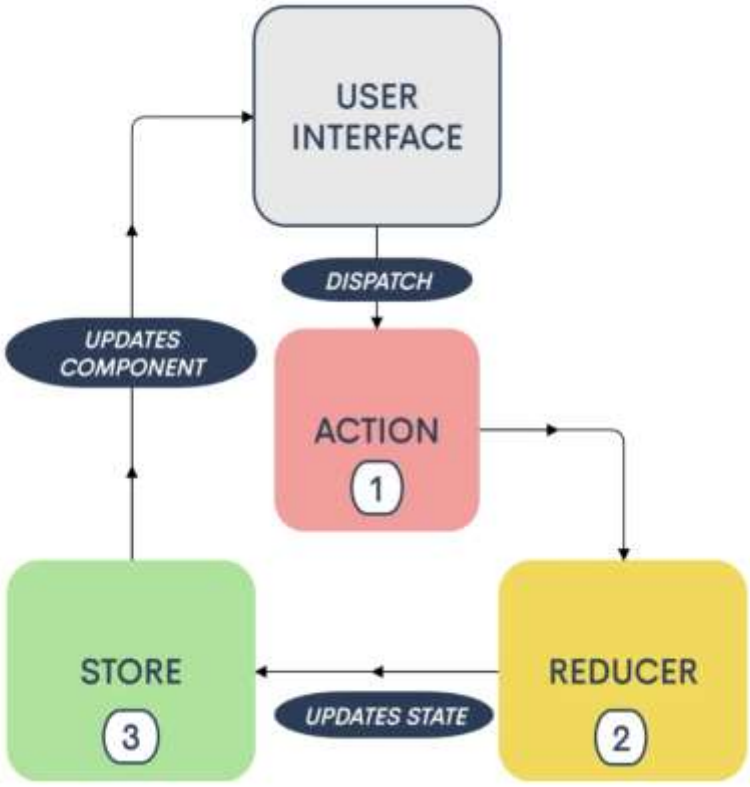

Рис. 1. Схема «Redux flow»

Однією з основних переваг Redux є його передбачуваність і структурованість. Усі зміни стану проходять через однаковий потік і визначений ланцюжок дій. Це дозволяє чітко відслідковувати зміни та уникати небажаних побічних ефектів. Додаткові розширення, такі як Redux DevTools, забезпечують прозорість і зручність для візуального відслідковування змін стану на етапі розробки.

Однак, використання Redux може призвести до збільшення обсягу коду через необхідність створення додаткових структур, таких як дії (actions) та редуктори (reducers). Крім того, часто використовуються допоміжні функції, хуки та проміжне програмне забезпечення (middleware) для роботи з API та асинхронними

операціями, наприклад, redux-thunk або redux-saga [24].

Якщо мова йде про додатки, написані з використанням TypeScript (TS), ситуація ускладнюється. Сьогодні важко уявити використання JavaScript (JS) без TS, оскільки останній поступово витісняє JS. Лише деякі додатки, здебільшого ті, що були розроблені давно і ще не трансформовані на TS, продовжують функціонувати на JS. Враховуючи це, кількість шаблонного коду в додатках на TS буде ще більшою, оскільки для використання усіх переваг і можливостей TS необхідно типізувати всі структурні одиниці в нашому сховищі [17].

З іншого боку, Context API, який є частиною стандартного набору інструментів React, пропонує простий і зручний підхід до управління станом (рис. 2). Він дозволяє уникнути необхідності встановлення сторонніх пакетів, надаючи можливість створювати контексти, які доступні будь-якому компоненту в дереві компонентів, подібно до Redux. Це допомагає уникнути передачі props через численні рівні ієрархії. Завдяки цим властивостям, Context API є ідеальним вибором для невеликих додатків або для управління окремими частинами стану, де централізована система не є необхідною. Context API забезпечує простий та інтуїтивно зрозумілий інтерфейс, що дозволяє швидко налаштувати систему управління станом [20].

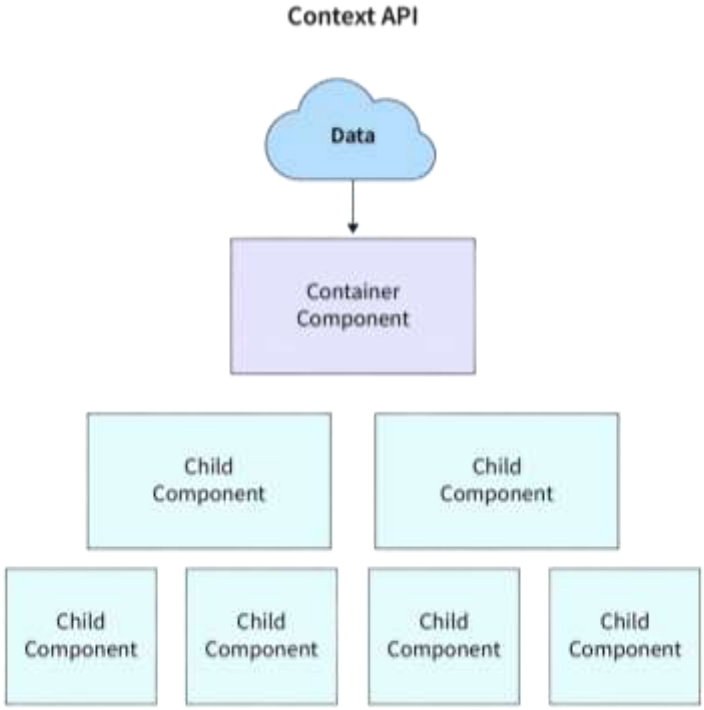

Рис. 2. Схема «Context API flow»

Проте зі зростанням складності додатка використання Context API може призвести до проблем з продуктивністю. Це особливо актуально, якщо мало уваги приділяється оптимізаційним аспектам, таким як використання хуків запам'ятовування (memoization), наприклад, useCallback та useMemo, або застосування кешування (caching) [16]. Проблема продуктивності здебільшого пов'язана зі збільшенням кількості рендерів компонентів. Зміни стану в контексті викликають повторний рендеринг усіх компонентів, які використовують цей контекст. Це, в свою чергу, негативно впливає на загальну продуктивність додатка.

Аналіз останніх джерел

З розвитком технологій веб-розробки питання забезпечення інтерактивності та динамічності інтерфейсів набуло особливої актуальності. Ефективне управління станом є однією з ключових задач, яка визначає спосіб зберігання, оновлення та передачі даних між компонентами. Враховуючи це, розробники створили безліч різних підходів та інструментів для таких цілей. Серед них, Redux і Context API, що є одними з найпоширеніших рішень і використовуються для організації та контролю стану. Кожен із цих підходів має свої переваги та недоліки [27].

Redux, розроблений Деном Абрамовим та Ендрю Кларком у 2015 році, швидко став стандартом де-факто для управління станом у великих та складних React-додатках. Його основні принципи базуються на концепції однонаправленого потоку даних, яка була вперше запропонована у Flux-архітектурі. Redux дозволяє зберігати весь стан додатку в одному глобальному сховищі, що робить його доступним для будь-якого компонента без необхідності передавати дані через ієрархію компонентів [12].

Згідно з численними дослідженнями, головними перевагами Redux є передбачуваність стану, яка забезпечується жорсткими правилами його зміни. Крім того, Redux надає можливість легко відслідковувати всі модифікації за допомогою інструментів, таких як Redux DevTools. Завдяки цим особливостям, він є ідеальним рішенням для застосунків, де важливо забезпечити узгодженість і контроль над усим додатком [17, 23].

З впровадженням React Hooks у версії 16.8, Context API набув нового значення як інструмент для управління станом у невеликих і середніх за розміром додатках. Context API відкриває можливість створення

контекстів, доступних для будь-якого компонента у дереві компонентів, що дозволяє уникнути передачі даних через численні рівні ієрархії [30]. Однією з ключових переваг Context API є його простота і зручність у використанні. Відсутність залежностей від зовнішніх бібліотек і швидкість налаштування системи управління станом роблять цей підхід все більш популярним серед команд розробників. Однак, зі зростанням складності додатку у випадках, коли багато компонентів підписані на зміни в одному контексті, виникає зайвий рендеринг, що негативно впливає на загальну продуктивності додатку. Це свідчить про те, що Context API має свої межі ефективності, особливо в умовах зростаючої складності та розширення функціональності проєкту [31, 32].

На основі аналізу літературних джерел, можна зробити висновок, що обидва підходи мають свої переваги та недоліки, і вибір між ними залежить від конкретних вимог проєкту. Redux є потужним інструментом для великих додатків, де необхідно управляти складною логікою та забезпечувати централізоване управління станом. Context API, навпаки, пропонує простий та зручний підхід для невеликих проєктів, де простота використання і швидкість розробки є ключовими факторами [5, 6].

Основна проблема, яку необхідно вирішити у даному дослідженні, полягає у визначенні доцільності застосування кожного з цих підходів залежно від обсягу проєкту, складності логіки та вимог до продуктивності. Це дослідження має на меті надати розробникам чіткі рекомендації щодо вибору інструментів для управління станом у сучасних React-додатках [8, 9].

У цьому дослідженні здійснено ґрунтовний порівняльний аналіз підходів Redux і Context API. Особливу увагу приділено їх продуктивності, зручності використання та можливості масштабування. На основі проведеного аналізу розроблено рекомендації щодо найбільш доцільного застосування кожного з підходів, враховуючи специфіку проєктів різного масштабу та складності.

Виклад основного матеріалу

Метою дослідження є глибоке вивчення та критичний аналіз Redux та Context API до управління станом у React-додатках. Основна мета полягає у визначенні їхніх переваг та обмежень в контексті різних аспектів розробки. Особлива увага приділяється практичній доцільності застосування цих інструментів в залежності від специфічних умов та вимог конкретних проєктів. Завданням дослідження є розробка рекомендацій для розробників щодо оптимального вибору засобів управління станом, що враховує особливості та масштаби проєкту.

Завдання дослідження спрямовані на досягнення кількох важливих цілей. Воно передбачає систематичний огляд та критичний аналіз існуючих наукових джерел і актуальних досліджень, що стосуються методів управління станом у React-додатках. Особлива увага приділяється таким інструментам, як Redux та Context API. Дослідження включає розробку та тестування прототипів додатків, побудованих з використанням обох підходів. Це дозволяє не тільки оцінити їх ефективність та продуктивність, але й проаналізувати масштабованість у реальних умовах. Проведено детальний порівняльний аналіз результатів експериментів, зосереджений на таких аспектах, як час рендерингу, споживання пам’яті та частота зайвих перерендерів компонентів. Крім того, дослідження оцінює складність імплементації кожного підходу залежно від розміру додатку та рівня гнучкості, необхідного для керування станом. На основі отриманих результатів, буде сформульовано практичні рекомендації щодо оптимального вибору інструменту для управління станом у залежності від специфіки проєкту, складності його логіки та вимог до масштабованості.

Для досягнення цілей дослідження було проведено поглиблене дослідження наукових статей, публікацій та технічної документації, присвячених управлінню станом у React-додатках [2–4]. Особлива увага була приділена порівнянню ефективності використання Redux і Context API в різних умовах, а також їх впливу на продуктивність і масштабованість додатків. Цей аналіз дозволив визначити ключові критерії для подальших експериментальних досліджень.

Для практичної перевірки отриманих даних була розроблена серія React-додатків з використанням обох підходів – Redux та Context API [13, 14]. У цих додатках було реалізовано типові сценарії, характерні для реальних проєктів: управління глобальним станом, а також взаємодії між компонентами. Продуктивність кожного підходу оцінювалась за допомогою таких метрик, як час рендерингу та обсяг спожитої пам'яті та інші показники, важливі для оцінки ефективності.

Було проведено детальний аналіз складності імплементації обох підходів, враховуючи розмір додатку та необхідний рівень гнучкості у керуванні станом. Зокрема, оцінювались обсяги коду, необхідні для налаштування і коректної роботи, кількість сторонніх залежностей, а також складність підтримки додатка в довгостроковій перспективі. Результати цього аналізу були систематизовані та представлені у вигляді таблиць, що відображають основні відмінності між підходами.

У рамках дослідження було проаналізовано продуктивність двох підходів до управління станом у простих і складних React-додатках, які виконують типові завдання, такі як зміна теми та мовних налаштувань. Для оцінки ефективності було використано Context API та Redux.

За результатами тестування, середній час рендерингу компонентів при використанні Context API виявився нижчим для невеликих проєктів. Це особливо актуально, якщо при оновленні стану не виконуються складні обчислення або обробка великих обсягів даних. У таких випадках Context API є достатньо швидким і ефективним рішенням. Використання Redux зазвичай є надмірним для таких задач. Крім того, якщо ієрархія компонентів у застосунку не дуже глибока, кількість компонентів, що підписані на зміни контексту, невелика, і його стан не потребує складної логіки управління, то Context API забезпечує кращу продуктивність (Табл.1).

Однак слід зазначити, що коли один і той самий контекст використовується для зберігання різних частин

стану чи даних, особливо якщо їхня логіка ніяк не пов'язана між собою, виникає проблема зайвого рендерингу. У такому випадку кожна зміна в контексті призводить до рендерингу всіх компонентів, які залежать від цього контексту, навіть якщо вони використовують лише частину цих даних. Це можна змінити методом розбиття одного великого контексту на декілька менших, щоб кожен контекст відповідав лише за одну групу пов'язаних даних.

Невеликий React-додаток складається з декількох сторінок, таких як: домашня сторінка, сторінка з контактами та сторінка з інформацією про компанію. В цьому додатку є можливість перемикання між світлою і темною темами, а також зміни мови інтерфейсу. Кількість компонентів обмежена, структура проста, і немає складних взаємозалежностей між компонентами. Користувач відкриває додаток і перемикає тему з темної на світлу. Додаток миттєво відображає зміни в стилях інтерфейсу. Після цього користувач змінює мову інтерфейсу з англійської на українську, і додаток знову швидко оновлює текст на екрані, відображаючи його українською мовою. В додатку використовується контекст для збереження стану теми. Цей контекст надає доступ до поточної теми всім компонентам, які цього потребують. Кореневий компонент обгорнутий у ThemeContext.Provider, який передає значення поточної теми всім дочірнім компонентам. Коли користувач змінює тему, значення в контексті оновлюється, і всі підписані компоненти автоматично перемальовуються. Подібним чином, LanguageContext зберігає стан мови. При зміні мови, контекст оновлюється, і всі компоненти, які використовують цей контекст, перемальовуються з оновленим текстом. У простому додатку, де компоненти розміщені на кількох рівнях ієрархії, Context API працює ефективно. Зміна теми чи мови викликає мінімальні зміни у стані, а кількість рендерингів є допустимою.

Таблиця 1

Результати порівняння часу рендерингу компонентів при виконанні різних дій у простих React-додатках з використанням Context API та Redux

Дія	Context API	Redux
Зміна теми (світла/темна)	47 ms	62 ms
Зміна мови інтерфейсу	41 ms	58 ms

Завдяки простоті контексту та відсутності необхідності в додаткових інструментах, Context API забезпечує швидку і плавну роботу додатка. Компоненти, які не залежать від теми чи мови, не оновлюються, що знижує навантаження на систему.

У свою чергу, складний додаток майже корпоративного рівня, окрім функцій перемикання теми та зміни мови інтерфейсу, включає в себе багато сторінок, складні компоненти з різними рівнями вкладеності та численні залежності. Додатково, у ньому присутня значна кількість інших функцій, таких як управління користувачами, аналітика, динамічні форми тощо. Користувач відкриває додаток і, як у попередньому сценарії, перемикає тему з темної на світлу та змінює мову інтерфейсу. Проте в цьому випадку застосунок має значну кількість компонентів, які реагують на зміни теми і мови. Ці компоненти розташовані на різних рівнях ієрархії, і майже кожен з них має залежність від теми або мови. У ньому стан теми зберігається в Redux store. Коли користувач змінює тему, виконується диспатч відповідної дії, яка оновлює стан теми в store. Використовуючи селектори, Redux дозволяє оновлювати лише ті компоненти, які залежать від теми, що значно знижує кількість перерендерів. Аналогічно, стан мови також зберігається в Redux store. Зміна мови викликає оновлення тільки тих компонентів, які потребують перерендерингу, завдяки селекторам, що дозволяє уникнути зайвих оновлень компонентів. У великому додатку використання одного загального контексту за допомогою Context API, без поділу на спеціалізовані контексти, призводить до зайвого рендерингу багатьох компонентів. Це відбувається тому, що зміни в контексті поширюються на всі підписані компоненти, навіть якщо вони використовують лише частину даних з цього контексту. Така ситуація створює додаткове навантаження на систему і знижує продуктивність. Redux, навпаки, завдяки централізованому управлінню станом та використанню селекторів, дозволяє оновлювати тільки ті компоненти, які безпосередньо залежать від зміненого стану.

Таблиця 2

Результати порівняння часу рендерингу компонентів при виконанні різних дій у складних React-додатках з використанням Context API та Redux

Дія	Context API	Redux
Зміна теми (світла/темна)	112 ms	93 ms
Зміна мови інтерфейсу	114 ms	101 ms

У великих додатках з багатьма компонентами та складною логікою Redux у порівнянні з одним контекстом з Context API діє більш ефективно. Особливо, якщо структура складна, а залежність між різними компонентами велика.

Таблиця 3

Оцінка ефективності Context API та Redux для різних типів проєктів

Тип проєкту	**Context API (Ефективність)**	**Redux (Ефективність)**
Невеликий проєкт	Висока	Середня
Середній проєкт	Середня	Висока
Великий проєкт	Низька	Висока

Таким чином, вибір між Context API та Redux повинен ґрунтуватись на складності проєкту або окремої його частини, кількості компонентів, які потребують управління станом, та вимогами до продуктивності. У простих застосунках Context API працює швидше завдяки своїй простоті, але у великих проєктах з багатьма взаємозалежними компонентами і станами Redux забезпечує кращу продуктивність в управлінні складною логікою.

Проведене нами дослідження охоплювало цілі системи, однак ці висновки також стосуються окремих частин програмного забезпечення, які можуть бути обгорнуті провайдерами Context API або Redux. Це означає, що не обов'язково обирати один підхід для всього проєкту. У великих і складних системах різні модулі можуть використовувати різні інструменти для управління станом, залежно від їхніх специфічних потреб (Табл.2). Наприклад, у межах одного об'ємного проєкту, деякі підсистеми або їх частини будуть ефективніше функціонувати з Context API, тоді як інші – з Redux, що дозволяє адаптувати управління станом під конкретні вимоги кожної частини програмного забезпечення (Табл.3).

Обговорення результатів

Отримані результати чітко демонструють, що вибір між Redux та Context API є багатофакторним процесом, який повинен базуватися на низці критеріїв, залежно від специфічних вимог кожного окремого проєкту або окремої частини його функціоналу. Основними параметрами, які слід враховувати при виборі відповідного інструменту, є розмір, складність логіки, потреба у масштабуванні, продуктивність, а також досвід і компетенції команди розробників.

За результатами аналізу, Redux вимагає більше пам'яті для управління складними додатками, проте забезпечує стабільність та передбачуваність оновлення стану (рис. 3). Context API, в свою чергу, демонструє кращі результати при роботі з невеликими додатками.

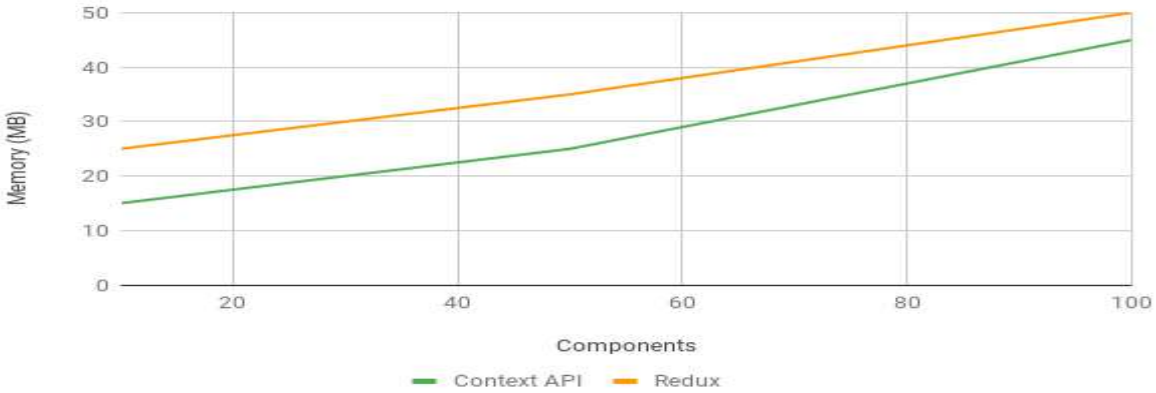

Рис. 3. Використання пам'яті у додатках з Redux та Context API

Для невеликих додатків або окремих частин великих проєктів, які мають просту структуру та відносно просту логіку, Context API є оптимальним вибором. Цей підхід забезпечує гарну продуктивність при мінімальних витратах ресурсів на налаштування та підтримку. Використання Context API дозволяє спростити архітектуру додатку, уникнути написання зайвого коду та встановлення сторонніх залежностей. В місцях, де необхідно управляти складною логікою та забезпечити централізоване управління станом, Redux є підходящим інструментом.

Один з основних аргументів на користь використання Redux полягає в його структурованості та передбачуваності. Використання єдиного сховища для управління станом дозволяє легко відслідковувати всі зміни, що відбуваються в додатку, та забезпечувати їх прозорість. Це особливо важливо в умовах роботи над великим проєктом. Крім того, Redux має добре розвинену екосистему, яка включає різноманітні інструменти та бібліотеки для роботи з асинхронними діями, оптимізації продуктивності та покращення розробницького досвіду. Це дозволяє розширювати функціональні можливості Redux залежно від потреб проєкту та вимог замовника. Важливим аспектом є також наявність потужних інструментів для налагодження, таких як Redux DevTools, що спрощує процес діагностики та вирішення проблем.

Висновки

Дослідження показало, що обрання між Redux та Context API може здатися складним завданням, але якщо розуміти їхні цілі та призначення, зробити правильний вибір стає значно простіше. Важливо не сприймати ці інструменти як взаємозамінні, а радше як такі, що доповнюють один одного. Кожен з них був розроблений для певних цілей, і, у випадку їх правильного поєднання, можна отримати оптимальне рішення для управління станом у проєкті.

Context API дозволяє уникнути "пропс-дрилінгу" і забезпечує просту та зручну систему управління станом у невеликих додатках або окремих частинах великих проєктів. Хоча й за допомогою нього можна повторити логіку поведінки Redux, принаймні спробувати її імітувати, створивши один глобальний контекст, такий підхід є неправильним, оскільки це призведе до зайвих перерендерів компонентів. Оптимальним рішенням є використання кількох невеликих контекстів, кожен з яких відповідає за певну частину логіки, що дозволяє уникати зайвих перерендерів і підвищувати ефективність додатку. Context API можна ефективно застосовувати для управління станом окремих модулів у великому проєкті при розбитті його на менші частини, кожна з яких має свій власний контекст. Це знижує навантаження на систему і підвищує загальну продуктивність.

З іншого боку, Redux краще підходить для централізованого управління глобальним станом, який покриває весь проєкт. Його використання особливо виправдане в складних і масштабних проєктах, де необхідно забезпечити передбачуваність і контроль над змінами стану. Використання Redux для простих завдань, таких як зміна теми чи мови інтерфейсу, не завжди доцільна, оскільки це вимагає додаткових зусиль на встановлення, налаштування та підтримку бібліотеки.

Вибір між Redux та Context API є багатовимірним процесом, який залежить від низки критеріїв, таких як масштаб, складність логіки, потреба у масштабуванні, продуктивність та досвід команди розробників. Проведене дослідження, хоча й охоплює цілі системи, стосується також окремих частин програмного забезпечення, які можуть бути обгорнуті провайдерами Context API або Redux. Це означає, що не завжди варто обирати один підхід для всього проєкту. У великих і складних системах різні модулі можуть використовувати різні інструменти для управління станом, залежно від специфічності вимог кожної частини. Розробники повинні ретельно аналізувати вимоги як до всього проєкту, так і до його окремих частин, щоб обрати найбільш підходящий інструмент, забезпечуючи баланс між простотою, ефективністю та функціональністю.

References

1. Abramov, D., & Clark, A. (2015). Redux: Predictable state container for JavaScript apps. GitHub. https://github.com/reduxjs/redux

2. MobX Documentation. MobX: Simple, scalable state management. https://mobx.js.org/

3. Alpert, B., Valk, L., & Grubbs, J. (2013). React: A JavaScript library for building user interfaces. Facebook Inc. https://reactjs.org/

4. Ng, R. (2017). State management in React applications. Medium. https://medium.com/

5. Kent, D. (2018). Redux vs Context API. Dev.to. https://dev.to/

6. Zakas, N. C. (2015). Understanding ECMAScript 6: The definitive guide for JavaScript developers. No Starch Press.

7. Stoyanovich, A. (2020). Managing state in React apps: A comparison between Redux and Context API. Smashing Magazine. https://www.smashingmagazine.com/

8. Hancock, T. (2019). Optimizing React performance with Redux. LogRocket Blog. https://blog.logrocket.com/

9. Gao, Q., & Lin, H. (2020). Advanced state management techniques in React. IEEE Xplore. DOI: 10.1109/JSAC.2020.296

10. Hartig, M., & Lieberman, J. (2017). React Architecture Best Practices. Apress. ISBN: 978-1484231342.

11. Huang, Z. (2018). Asynchronous state management with Redux and redux-thunk. Dev.to. https://dev.to/

12. Lauer, T. (2016). Exploring the Redux middleware. SitePoint. https://www.sitepoint.com/

13. Cheng, J. (2021). State management in React apps: Tips and tricks. Stack Overflow. https://stackoverflow.blog/

14. McCartney, J. (2019). The evolution of state management in JavaScript. CSS-Tricks. https://css-tricks.com/

15. Silva, R. (2019). Choosing the right state management library for your React application. Dev.to. https://dev.to/

16. Garcia, L. (2018). Redux vs. MobX: Which state management library to choose?. Telerik. https://www.telerik.com/

17. Pedersen, K. (2020). React Context API vs Redux: Pros and cons. Hacker Noon. https://hackernoon.com/

18. Rosca, D. (2021). State management in React: A practical guide. Udemy. https://www.udemy.com/

19. Boakye, E. (2020). Managing complex state in React applications with Redux. FreeCodeCamp. https://www.freecodecamp.org/

20. Devito, M. (2019). Improving React application performance with Context API. Stack Overflow. https://stackoverflow.blog/

21. Lee, H. (2018). State management with Redux in large React applications. The Startup.

https://medium.com/

22. Walsh, C. (2021). The future of state management in React. The Web Dev. https://thewebdev.info/
23. Jennings, T. (2020). Understanding the React Context API. Medium. https://medium.com/
24. Naimark, D. (2019). Choosing between Redux and Context API. Dev.to. https://dev.to/
25. Elmasri, R. (2020). React performance optimization: Redux vs Context API. Smashing Magazine. https://www.smashingmagazine.com/
26. Walsh, P. (2021). React Context API: Use cases and best practices. Telerik. https://www.telerik.com/
27. Black, S. (2020). Advanced Redux techniques for scalable React applications. SitePoint. https://www.sitepoint.com/
28. Jimenez, P. (2019). How to optimize React apps with Context API. LogRocket Blog. https://blog.logrocket.com/
29. Wu, C. (2021). State management in modern React applications: A comparative study. IEEE Access. DOI: 10.1109/ACCESS.2021.308
30. Gupta, A. (2020). Redux or Context API: Which one to use in React projects?. Medium. https://medium.com/
31. Rivera, L. (2021). React performance optimization with Redux and Context API. The Startup. https://medium.com/
32. Pereira, D. (2021). Understanding Redux and Context API for state management. SitePoint. https://www.sitepoint.com/
33. Anderson, M. (2019). React state management: A comprehensive guide. Packt Publishing. ISBN: 978-1838981732.
34. Evans, S. (2020). Context API vs Redux: A performance analysis. JavaScript Weekly. https://javascriptweekly.com/
35. Brown, J. (2021). Advanced patterns in Redux and Context API. Smashing Magazine. https://www.smashingmagazine.com/