ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ

StrikePlagiarism.com

Дата звіту 5/24/2025
Дата редагування —

Звіт не був оцінений

Звіт подібності

метадані

Назва організації
Kharkiv National University of Radio Electronics
Заголовок
2025_М_ПІ_ІПЗздм-23-1_Малікова_Т_В_скорочений.pdf
Автор Науковий керівник / Експерт
Малікова Тетяна ВіталіївнаСмеляков К.С./Вадим Юрійович Нчволод
підрозділ
каф. ПІ

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

0.27% 0.27% КП 1

0.27% 0.27% КЦ

25	8517	69328
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		1
Інтервали		0
Мікропробіли		0
Білі знаки		0
Парафрази (SmartMarks)		2

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Розроблення вебзастосунку "Галерея 3D моделей" [redacted] 6/7/2023 National Forestry University of Ukraine (ЦДН НЛТУ України)	6 0.07 %
2	https://ir.nmu.org.ua/jspui/bitstream/123456789/161771/1/122-18-1_%D0%91%D0%B8%D1%87%D0%BA%D0%BE%D0%B2%20%D0%86.%D0%9E.pdf	6 0.07 %
3	https://ir.nmu.org.ua/jspui/bitstream/123456789/161771/1/122-18-1_%D0%91%D0%B8%D1%87%D0%BA%D0%BE%D0%B2%20%D0%86.%D0%9E.pdf	6 0.07 %

4	Розроблення вебзастосунку "Галерея 3D моделей" [redacted] 6/7/2023 National Forestry University of Ukraine (ЦДН НЛТУ України)	5 0.06 %

з бази даних RefBooks (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з домашньої бази даних (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з програми обміну базами даних (0.13 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Розроблення вебзастосунку "Галерея 3D моделей" [redacted] 6/7/2023 National Forestry University of Ukraine (ЦДН НЛТУ України)	11 (2) 0.13 %

з Інтернету (0.14 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://ir.nmu.org.ua/jspui/bitstream/123456789/161771/1/122-18-1_%D0%91%D0%B8%D1%87%D0%BA%D0%BE%D0%B2%20%D0%86.%D0%9E.pdf	12 (2) 0.14 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)

12

ВСТУП

Сучасні односторінкові застосунки (Single Page Applications, SPA) є невід'ємною складовою сучасної веб-розробки, яка стрімко розвивається у відповідь на потреби користувачів у швидких, зручних та інтерактивних інтерфейсах. Такі застосунки дозволяють значно зменшити обсяги даних, які передаються між клієнтом і сервером, завдяки завантаженню єдиної HTML-сторінки та динамічному оновленню вмісту через JavaScript. Це забезпечує високу продуктивність, кращу взаємодію з користувачем та зменшення часу очікування при навігації між сторінками.

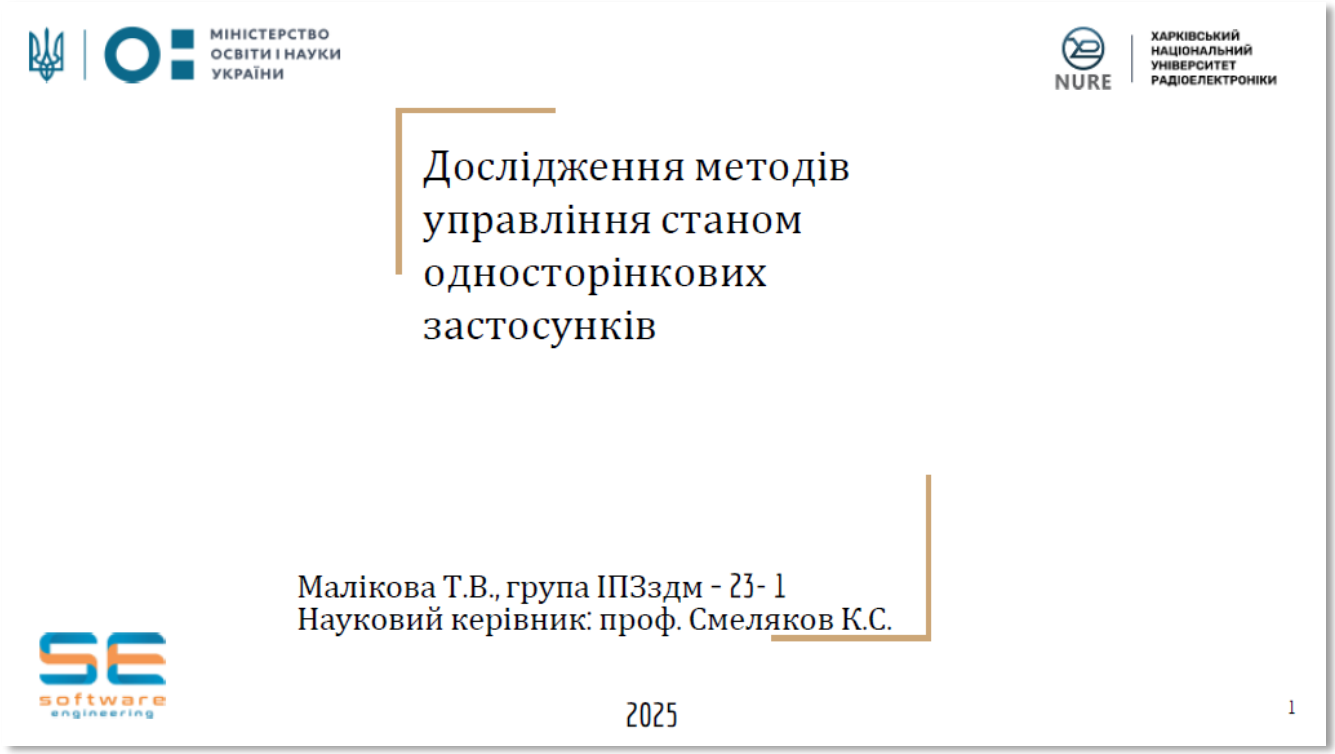
Разом з тим, збільшення обсягів функціональності, інтеграція сторонніх сервісів, ускладнення логіки взаємодії між компонентами та посилення вимог до масштабованості призводять до серйозних викликів у сфері управління станом. Стан застосунку – це сукупність усіх даних, які відображають поточний стан інтерфейсу, користувацького вводу, API-викликів тощо. Його ефективна організація та підтримка є критично важливими для стабільності та підтримуваності проєкту.

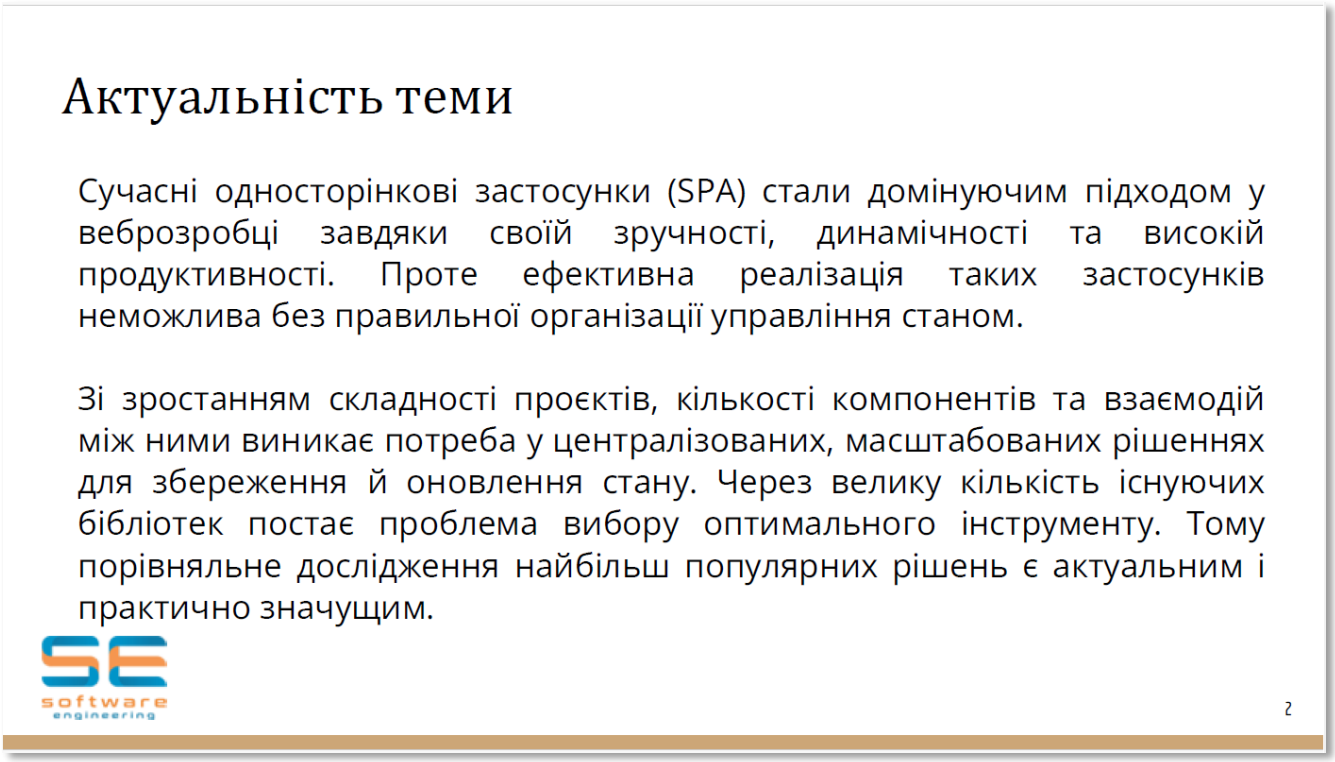
Фреймворк React, один із найпопулярніших інструментів для створення SPA, надає кілька базових засобів для управління станом, таких як хуки useState, useReducer, а також Context API для передачі стану між компонентами. Проте у реальних проєктах цього часто недостатньо, особливо якщо застосунок має складну архітектуру або велике дерево компонентів. У таких випадках розробники вдаються до використання сторонніх бібліотек для централізованого управління станом.

Серед найпоширеніших рішень виділяються Redux та Zustand. Redux є надійним та широко застосовуваним інструментом з чіткою структурою та підтримкою великої екосистеми. Він базується на суворій односпрямованій передачі даних та використанні єдиного сховища (store), що дозволяє контролювати всі зміни стану в одному місці. Водночас, Redux потребує написання значного обсягу шаблонного коду, що може ускладнити розробку.

ДОДАТОК Б

Слайди презентації





Аналіз предметої галузі

SPA (Single Page Application) - веб-додатки, які оновлюють контент без повного перезавантаження сторінки, покращують користувацький досвід завдяки швидкому реагуванню.

Проблеми управління станом у SPA полягають у необхідності відстежувати зміни в багатьох компонентах, асинхронності і потребі у глобальному стані.

Фреймворк React - декларативний підхід до побудови UI, компонентна архітектура для розподілу логіки.

Головним **викликом** є те, що локальний стан підходить тільки для простих завдань, а глобальний стан вимагає спеціальних рішень.

Об'єкт і предмет дослідження

Об'єкт дослідження — методи управління станом в односторінкових застосунках на базі фреймворку React.

Предмет дослідження — порівняння роботи конкретних бібліотек: Redux, Zustand, Recoil і Context API.

У центрі уваги — принципи організації стану, способи передачі даних, вплив на рендеринг компонентів, продуктивність, легкість реалізації та масштабованість проєктів.

Постановка задачі

Мета: провести теоретичний та експериментальний аналіз методів глобального управління станом, визначити найефективніші підходи для застосування в SPA-застосунках.

Завдання:

- Описати проблему керування станом у React.
- Провести огляд і класифікацію популярних бібліотек.
- Розробити прототипи застосунків з однаковою логікою, використовуючи різні бібліотеки.
- Провести заміри продуктивності, обсягу коду, кількості ререндерів.
- Побудувати візуалізації результатів і сформулювати практичні висновки.

Теоретична база дослідження

В основу дослідження покладено:

- офіційну документацію бібліотек (React, Redux, Zustand, Recoil),
- фахову літературу (посібники, книги з управління станом у React),
- статті з технічних блогів та порталів (Medium, Dev.to),
- навчальні ресурси (Redux Toolkit, React Patterns),
- практичний досвід розробників у відкритих проєктах.

Аналіз джерел дозволив сформувати чітке розуміння різних стратегій роботи зі станом.

Проблема локального управління станом

Локальні засоби React (useState, useReducer) ефективні лише у невеликих компонентах або малих проєктах. Зі зростанням дерева компонентів:

- зростає кількість "прокидування" даних (проп-дрилінг),
- важче синхронізувати зміни між компонентами,
- рендеринги відбуваються частіше, що впливає на продуктивність,
- ускладнюється підтримка архітектури.

Для уникнення цих проблем потрібен перехід до глобального стану.

Redux: масштабованість і структура

Redux — одна з найстаріших і найпопулярніших бібліотек для глобального керування станом.

Принципи:

- Централізоване store.
- Зміни здійснюються через actions і обробляються reducers.
- Висока передбачуваність, чітка архітектура.

Переваги:

- Добре працює у великих проєктах,
- підтримка DevTools,
- зручність тестування та масштабування

Недоліки:

- Шаблонний код,
- складна інтеграція.

Zustand: мінімалізм і ефективність

Zustand — сучасна легка бібліотека, яка:

- використовує функціональний стиль створення глобального стану,
- не потребує Provider,
- має простий API,
- дозволяє вибіркову підписку на частини стану (селектори).

Переваги:

- Висока продуктивність.
- Мінімум коду.
- Зручна інтеграція.
- Ідеально підходить для невеликих і середніх застосунків.

Context API: базове глобальне рішення

Context API — вбудований механізм React для передачі стану без проп-дрилінгу.

Особливості:

- Створення Context,
- передача через Provider,
- споживання через useContext.
- Добре працює з невеликими наборами даних або глобальними параметрами (тема, мова, користувач).

Обмеження:

- немає автоматичної оптимізації ререндерів,
- відсутні засоби для складної логіки.

Recoil: атомарна структура стану

Recoil — бібліотека від Meta, орієнтована на складні взаємозалежності стану.

Основи:

- atoms — базові одиниці стану.
- selectors — похідні значення на базі атомів.

Переваги:

- Гнучка організація складного стану.
- Гарна інтеграція з React.
- Добре підходить для модульних, ізольованих компонентів.

Обмеження:

- бібліотека нова,
- розвивається,
- менше документації.

11

Теоретичний порівняльний аналіз

На основі аналізу документації, оглядів та прикладів реалізації:

Критерій	Context API	Redux	Zustand	Recoil
Продуктивність	Середня	Висока	Висока	Висока
Простота	Висока	Низька	Дуже висока	Висока
Масштабованість	Обмежена	Висока	Висока	Висока
Передбачуваність	Середня	Висока	Висока	Висока
Документація	Стандартна	Розширена	Середня	Обмежена

12

Експериментальна частина дослідження

Для перевірки теоретичних висновків було розроблено чотири версії SPA:

- однакова логіка — бронювання місць,
- різні реалізації стану: Redux, Zustand, Context API, Recoil,
- однакова структура, верстка, інтерфейс.

Це дозволило провести об'єктивне порівняння на основі реального функціоналу.

Реалізація з Redux

•Окремий модуль seatsSlice.js з редюсерами та екшенами.

•Стан зберігається в централізованому сховищі.

•Компоненти отримують доступ через useSelector і useDispatch.

•Додаткові можливості: middleware, DevTools.

•Структура коду складна, але надійна.

Реалізація з Zustand

•Створено store за допомогою create().

•Компоненти підписуються лише на потрібні частини стану.

•Рендер відбувається лише для тих компонентів, які потребують оновлення.

•Код — мінімалістичний, без шаблонних структур.

Реалізація з Context API

•Створено SeatsContext та обгортка Provider.

•Компоненти читають/оновлюють стан через useContext.

•Контекст оновлює всі підписані компоненти.

•Важко масштабувати при великій кількості змін.

Реалізація з Recoil

•Створено atom для глобального стану місць.

•selectors використовуються для агрегаційної логіки.

•Компоненти використовують useRecoilState, useRecoilValue.

•Висока гнучкість і контроль залежностей.

Цілі експерименту та параметри порівняння

Ключова мета — перевірити ефективність кожного підходу в реальних умовах.
Зібрані метрики:

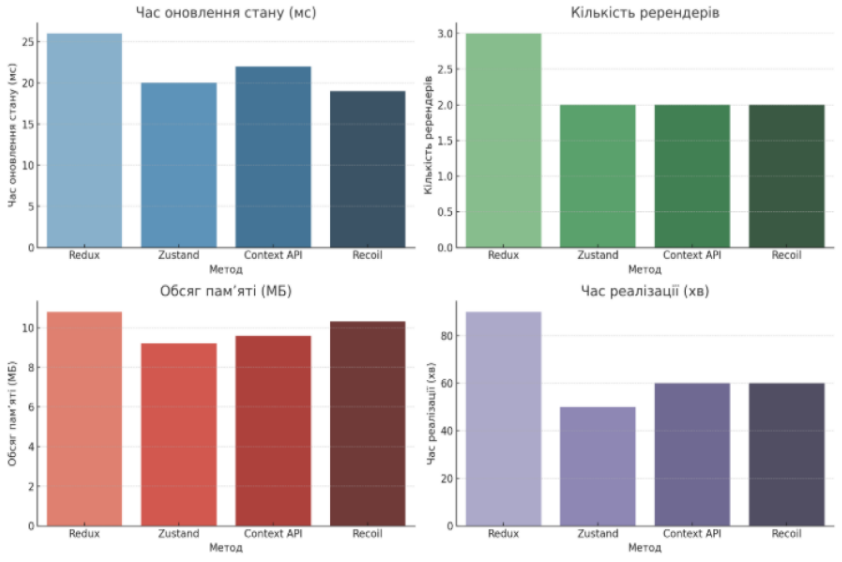
- час оновлення стану (мс),
- кількість повторних ререндерів,
- споживання пам’яті (МБ),
- обсяг коду (рядки),
- час реалізації (хв),
- експертна оцінка масштабованості (1–5).

Використано Chrome DevTools, React Profiler, статистичний аналіз результатів.

Таблиця результатів вимірювань

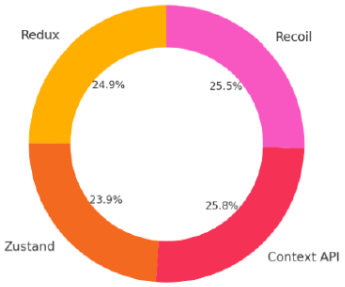
Метод	Час оновлення стану (мс)	К-сть ререндерів	Обсяг пам'яті (МБ)	Обсяг коду (рядки)	Час реалізації (хв)	Масштабованість (1–5)
Redux	26	3	10.8	169	90	5
Zustand	20	2	9.2	162	50	4
Context API	22	2	9.6	175	60	2
Recoil	19	2	10.3	173	60	4

Порівняння методів управління станом у React

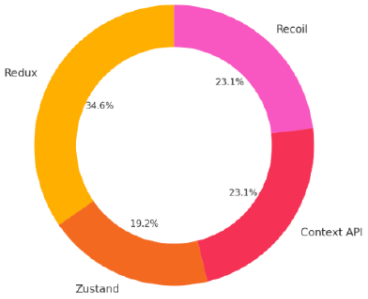


Розподіл обсягу коду та часу реалізації

Розподіл обсягу коду (рядки)

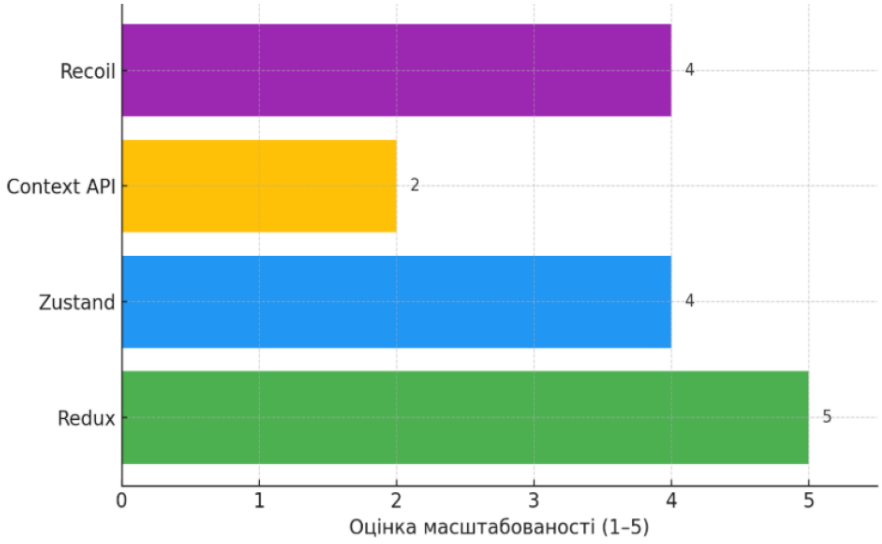


Розподіл часу реалізації (хв)



21

Рейтинг масштабованості



22

Інтерпретація результатів

Аналіз результатів експерименту показав, що бібліотеки Zustand і Recoil демонструють найкращі показники продуктивності: вони забезпечують швидке оновлення стану, мінімальну кількість рендерів і економне використання пам'яті. Context API показав себе добре у простих реалізаціях, але виявив обмеження при ускладненні архітектури. Redux продовжує залишатися стабільним вибором для великих застосунків, хоча його складність і обсяг коду значно перевищують альтернативні рішення.

Узагальнюючи результати, можна зробити висновок, що для невеликих або середніх проєктів доцільно використовувати Zustand або Recoil завдяки простоті впровадження та високій ефективності. Redux доцільно застосовувати у великих командах, де важливі передбачуваність і контроль над потоками даних. Context API варто використовувати лише для локальних або допоміжних задач. Отримані результати дозволили не лише кількісно порівняти бібліотеки, а й сформувати рекомендації щодо їх використання в різних умовах.

Висновки

У рамках дослідження було проаналізовано та порівняно методи управління станом у React-застосунках: Redux, Zustand, Context API та Recoil. Встановлено, що кожен підхід має свої переваги залежно від складності проєкту та вимог до продуктивності.

Redux забезпечує передбачуваність та масштабованість, однак потребує більше коду. Zustand виявився найпростішим у реалізації та досить ефективним. Context API придатний для невеликих застосунків, а Recoil — для проєктів зі складними залежностями.

Результати практичних вимірювань підтвердили теоретичні висновки, дозволивши сформувати обґрунтовані рекомендації щодо вибору бібліотек для управління станом у SPA.

ДОДАТОК В

Апробація результатів роботи

Дослідження Методів Управління Станом Односторінкових Застосунків

Тетяна Малікова[a], Кирило Смеляков[b]

Харківський національний університет радіоелектроніки, пр. Науки 14, м. Харків, 61166, Україна
Харківський національний університет радіоелектроніки, пр. Науки 14, м. Харків, 61166, Україна

Анотація

У статті проведено дослідження сучасних підходів до управління станом у фреймворку React, який широко використовується для створення односторінкових застосунків (SPA). Проаналізовано можливості та архітектурні особливості таких рішень, як Context API, Redux, MobX, Zustand і Recoil. Розглянуто переваги й обмеження кожного методу в контексті розробки масштабованих і продуктивних вебзастосунків, а також представлено результати практичного тестування їх ефективності.

Ключові слова 1

React, управління станом, SPA, Redux, MobX, Zustand, Recoil, односторінкові застосунки, Context API, useState, useReducer, глобальний стан, проп-дрилінг, веброзробка, фронтенд, продуктивність, масштабованість, JavaScript, сховище даних, actions, редюсери, атомарна модель, реактивне програмування, селектори, компонентна архітектура, вебзастосунки, передача даних, ієрархія компонентів, оптимізація продуктивності, архітектурні патерни, крос-компонентна комунікація, TypeScript, клієнтська розробка

1. Вступ

Односторінкові застосунки (SPA) стали основним підходом у розробці сучасних вебсервісів завдяки своїй інтерактивності, швидкому оновленню інтерфейсу та плавному користувацькому досвіду [1]. Разом з тим, розробка таких застосунків супроводжується складністю ефективного управління станом, особливо коли йдеться про проєкти з великою кількістю компонентів і взаємозалежностей [7] [9].

Фреймворк React, незважаючи на потужність внутрішнього механізму роботи з локальним станом, не завжди може забезпечити достатній рівень керованості при масштабуванні [1]. Саме тому в розробці з'являється потреба в застосуванні зовнішніх інструментів та бібліотек, що дозволяють централізовано контролювати стан, знижувати складність архітектури та покращувати продуктивність.

2. Основні підходи до управління станом у React

Локальний стан компонентів (useState, useReducer) є базовим механізмом у React, який забезпечує простоту використання [1]. Проте в умовах зростання проєкту та необхідності передавання даних між багатьма компонентами та рівнями вкладеності, цей підхід стає малоефективним. З'являється так званий "проп-дрилінг", що ускладнює підтримку та масштабування застосунку [7].

Для розв'язання цієї проблеми React пропонує Context API — вбудований механізм глобального стану, що дозволяє надавати доступ до даних на будь-якому рівні компонентної

MIT@AIS'2025s: 1st International Scientific and Practical Conference "Modern Information Technologies and Artificial Intelligence Systems", May 19–22, 2025, Kharkiv-Yaremche, Ukraine
EMAIL: tetiana.malikova.cpe@nure.ua (A. 1); kyrylo.smelyakov@nure.ua (A. 2)
ORCID: 0009-0009-4910-9895 (A. 1); 0000-0001-9938-5489 (A. 2)

ієрархії [1]. Проте при частих оновленнях стану контексту виникають проблеми з надмірними повторними рендерами, що впливають на продуктивність.

Redux є найпоширенішим інструментом для управління станом у великих застосунках [2]. Його архітектура базується на централізованому сховищі (store), що містить єдиний глобальний стан, а також суворій структурі оновлення стану через дії (actions) та редюсери [6]. Це забезпечує передбачуваність та можливість ефективного дебагінгу. Попри свої переваги, Redux потребує написання значного обсягу шаблонного коду, що може ускладнити вхід у проєкт для нових розробників [7].

MobX пропонує реактивний підхід, що автоматизує оновлення стану через спостереження за змінами [3]. Така модель дозволяє зосередитися на бізнес-логіці, мінімізуючи вручну керовані оновлення. Однак саме ця автоматизація може знижувати передбачуваність поведінки системи у великих проєктах [7].

Zustand — сучасна бібліотека, що надає мінімалістичний API і поєднує простоту з високою продуктивністю [4]. Вона дозволяє створювати глобальний стан без складної конфігурації. Завдяки відсутності шаблонного коду й ефективному механізму оновлення компонентів Zustand є зручною альтернативою для середніх проєктів.

Recoil, ще одне нове рішення від Meta, реалізує атомарну модель стану [5]. Це дозволяє точно контролювати залежності між частинами стану через атоми та селектори. Такий підхід забезпечує гнучкість і масштабованість, проте бібліотека ще перебуває у фазі активного розвитку.

3. Порівняльна оцінка бібліотек

У процесі дослідження було проведено порівняльний аналіз найбільш популярних бібліотек для управління станом у React-застосунках: Context API, Redux, MobX, Zustand та Recoil. Для об'єктивного порівняння були визначені ключові критерії оцінювання [7]:

- Продуктивність — швидкість оновлення інтерфейсу при зміні стану, кількість зайвих рендерів, вплив на ресурси системи.
- Простота використання — зручність інтеграції бібліотеки у проєкт, зрозумілість синтаксису та документації.
- Масштабованість — можливість застосування бібліотеки у великих проєктах з великою кількістю взаємодіючих компонентів.
- Передбачуваність та керованість — наскільки легко контролювати оновлення стану, відстежувати зміни, відлагоджувати логіку.
- Гнучкість — можливість роботи з асинхронними подіями, побудова складних залежностей між частинами стану.
- Підтримка спільноти та екосистеми — наявність розширень, популярність, активність підтримки та обсяги документації.

За результатами теоретичного аналізу та практичних експериментів було встановлено:

- Context API показав хорошу інтеграцію в простих проєктах. Він не потребує встановлення додаткових пакетів, має вбудовану підтримку в React. Проте при зростанні складності застосунку його ефективність знижується через перевантаження контексту та зайві перендери..
- Redux продемонстрував найбільшу передбачуваність та стабільність, завдяки суворій структурі потоків даних, централізованому сховищу та потужним DevTools. Бібліотека дозволяє реалізувати складні архітектури з великою кількістю компонентів, що взаємодіють. Недоліком є значний обсяг шаблонного коду, потреба у додаткових бібліотеках для роботи з асинхронністю (наприклад, redux-thunk або redux-saga), а також відносно висока складність налаштування [2][6].
- MobX запропонував реактивну модель, що автоматично відстежує залежності між станом і компонентами. Це дозволяє мінімізувати кількість рендерів і забезпечує високу продуктивність. MobX простий у впровадженні, але у великих проєктах може виникати втрата контрольованості через автоматичне поширення змін без централізованого керування [3].

• Recoil запропонував унікальний підхід — атомарну модель управління станом, що дозволяє точно визначати залежності та уникати зайвих оновлень. Це забезпечує масштабованість і контроль, однак бібліотека ще не така зріла, як Redux чи MobX, і потребує подальшої стабілізації та розвитку екосистеми [5].

• Zustand став компромісом між простотою та продуктивністю. Завдяки мінімалістичному API, простій інтеграції, підтримці селекторів та можливості гнучко організувати глобальний стан без провайдерів, Zustand показав себе як зручний та легкий інструмент для середніх і великих застосунків [4].

Таким чином, кожен з інструментів має свої сильні сторони та обмеження, що підтверджує необхідність врахування особливостей конкретного проєкту при виборі бібліотеки для управління станом.

4. Практичні результати

У рамках дослідження були реалізовані тестові проєкти, в яких використовувалися кожна з розглянутих бібліотек.

Порівняльна оцінка зазначених бібліотек здійснювалась за такими критеріями: продуктивність, простота використання, масштабованість, передбачуваність та підтримка спільнотою [7]. Результати свідчать, що Context API добре підходить для простих рішень, Redux — для великих, суворо структурованих проєктів, MobX — для швидкої розробки, Recoil — для складних систем із багатьма залежностями, а Zustand — для тих випадків, де важлива простота без втрати ефективності.

Таким чином, дослідження підтвердило, що вибір бібліотеки управління станом має залежати від складності та специфіки проєкту, вимог до продуктивності, можливості масштабування та досвіду команди. Context API показав хороші результати для невеликих застосунків, проте був обмежений у продуктивності при частих оновленнях стану [1]. Redux забезпечив найкращу стабільність і передбачуваність, хоча потребував значного обсягу коду [2] [6]. Zustand і Recoil виявилися найбільш збалансованими за критерієм простоти впровадження та ефективності [4] [5]. MobX забезпечив найвищу швидкість реалізації, але продемонстрував зниження контрольованості в умовах складної логіки [3].

5. Висновки

Управління станом у React-застосунках є критично важливим аспектом, особливо при розробці масштабованих SPA [7] [8]. Вибір конкретного методу має ґрунтуватися на складності проєкту, вимогах до продуктивності та потребах у підтримці. Універсального рішення не існує: Context API доречний для невеликих систем [1], Redux — для великих структурованих проєктів [2] [6], MobX — для швидкої розробки [3], Recoil — для роботи з залежностями [5], а Zustand — як легка та ефективна альтернатива [4].

Майбутні дослідження можуть бути зосереджені на гібридних моделях управління станом, поєднанні кількох бібліотек, а також на створенні власних інструментів, адаптованих до специфіки конкретних проєктів [7] [8] [9].

6. Література

[1] React.js Documentation. URL: https://reactjs.org/docs/
[2] Redux Documentation. URL: https://redux.js.org/
[3] MobX Documentation. URL: https://mobx.js.org/README.html
[4] Zustand Documentation. URL: https://docs.pmnd.rs/zustand/getting-started/introduction
[5] Recoil Documentation. URL: https://recoiljs.org/
[6] Abramov D., Clark A. Redux: Predictable State Container for JS Apps. GitHub, 2015. URL: https://github.com/reduxjs/redux
[7] Hall J. Mastering React State Management. Packt Publishing, 2022. 290 с.

[8] Cherny B. Learning TypeScript: Enhance Your Web Development Skills. O'Reilly Media. 2022. 324 c.
[9] Kirill, S., Pribylnov, D., Martovytskyi, V., Chupryna, A.14th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering, TCSET 2018 - Proceedings, 2018, 2018-April, pp. 983–986: Investigation of network infrastructure control parameters for effective intellectual analysis

Науковий керівник Смеляков Кирило Сергійович, доктор технічних наук, професор, кафедра Програмної Інженерії Харківського національного університету радіоелектроніки

ДОДАТОК Г

Експертний висновок результатів перевірки кваліфікаційної роботи на відповідність оформлення вимогам ДСТУ 3008: 2015

Експертний висновок результатів перевірки кваліфікаційної роботи

студент (посада) | програмної інженерії (кафедра) | ІПЗздм-23-1 (група)

Маліковій Тетяні Віталіївні
(прізвище, ім'я, по батькові)

Зауваження

Пункт ДСТУ 3008-2015	Зміст пункту	Сторінка кваліфікаційної роботи
1	2	3
	7.1 Загальні положення	
	7.3 Нумерація сторінок звіту	
	7.4 Нумерація розділів, підрозділів, пунктів, підпунктів	
	7.5 Рисунки	
	7.6 Таблиці	
	7.7 Переліки	
	7.8 Примітки	
	7.9 Виноски	
	7.10 Формули та рівняння	
	7.11 Посилання	
	7.13 Список авторів	
	7.14 Скорочення та умовні познаки	
	7.15 Додатки	

зауважень немає

Експерт _____ (підпис) Олена ОЛІЙНИК (прізвище, ініціали)

24.05.2025