Міністерство освіти і науки України
Харківський національний університет радіоелектроніки
Факультет Комп'ютерних наук
Кафедра Програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

другий (магістерський)
(рівень вищої освіти)

Дослідження швидкодії односторінкових застосунків в залежності від різних методів управління станом на прикладі React

Виконала:
Студентка 2 курсу групи ІПЗм-20-2
Проніна Д. М.
(прізвище, ініціали)
Спеціальність 121 — Інженерія програмного забезпечення
Тип програми Освітньо-наукова
Керівник ст.викл., к.т.н. Кириченко І. В.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. Кафедри _____ З. В. Дудар

2022

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Програмної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 121 — Інженерія програмного забезпечення

Спеціалізація Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. Кафедри _____

«_____» _____ 20___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентки Проніній Дар'ї Миколаївні
(прізвище, ім'я, по батькові)

1. Тема роботи: «Дослідження швидкодії односторінкових застосунків в залежності від різних методів управління станом на прикладі React»

затверджена наказом університету від « 24 » березня 2022 р.

2. Термін подання студенткою роботи до екзаменаційної комісії « 10 » травня 2022 р.

3. Вихідні дані до роботи *методи управління станом, OS MacOS, мова програмування JavaScript, бібліотека React, середовище розробки WebStorm.*

4. Перелік питань, що потрібно опрацювати в роботі *мета роботи, аналіз предметної галузі і постановка задачі, дослідження різних методів управління станом, зокрема їхній вплив на швидкодію односторінкових застосунків.*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	24.01.2022	виконано
2	Огляд існуючих методів управління станом в односторінкових React-застосунках	03.02.2022	виконано
3	Постановка задачі	14.02.2022	виконано
4	Вибір методів та критеріїв оцінки швидкої односторінкових застосунків	16.02.2022	виконано
5	Дослідження швидкодії односторінкових застосунків в залежності від різних методів управління станом	26.03.2022	виконано
6	Підготовка пояснювальної записки	13.04.2022	виконано
7	Перевірка роботи на антіплагіат	10.05.2022	виконано
8	Нормоконтроль	11.05.2022	виконано
9	Рецензування	13.05.2022	виконано
10	Підготовка презентації та доповіді	13.05.2022	виконано
11	Попередній захист	14.05.2022	виконано
12	Занесення роботи в електронний архів	14.05.2022	виконано
13	Допуск до захисту у зав. кафедри	19.05.2022	виконано

Дата видачі завдання « 24 » січня 202__ р.

Студентка _____
(підпис)

Керівник роботи _____ ст.викл., к.т.н. Кириченко І. В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Квалифікаційна робота магістра містить: 87 с., 14 рис., 6 табл., 33 джерел , 8 додатків.

ФРОНТ-ЕНД ТЕХНОЛОГІЇ, REACT, REDUX, ХУКИ, RECOIL, ШВИДКОДІЯ, ОДНОСТОРІНКОВІ ПРОГРАМИ

Об’єктом дослідження є швидкодія односторінкових застосунків в залежності від різних методів управління станом на прикладі React.

Метою роботи є порівняння швидкодії односторінкового застосунку з використанням React Hooks. Redux та Recoil у якості менеджерів стану.

В результаті роботи проведено аналіз подібних досліджень, був зроблений огляд найбільш популярних методів управління станом та експериментальне порівняння швидкодії односторінкового застосунку з використанням React Hooks, Redux та Recoil у якості менеджерів стану.

FRONT-END TECHNOLOGIES, REACT, REDUX, HOOKS, RECOIL, PERFORMANCE, SINGLE PAGE APPLICATIONS

The object of research is the performance of one-page applications depending on different methods of state management on the example of React.

The purpose of the work is to compare the performance of a one-page application using React Hooks, Redux and Recoil as state managers.

As a result, an analysis of similar studies was conducted, a review of the most popular state management methods and an experimental comparison of the performance of a one-page application using React Hooks, Redux and Recoil as state managers.

Я, Проніна Дар’я Миколаївна, студентка гр. ІПЗм-20-2, здобувачка вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження швидкодії односторінкових застосунків в залежності від різних методів управління станом на прикладі React», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлена з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

Створення інтерфейсу користувача розвивається так само, як і інші компоненти сучасних веб-систем. Одним із підходів до створення за допомогою інтерфейсів є реалізація веб-додатка як односторінкової програми, або SPA.

Серед популярних бібліотек для побудови користувацьких інтерфейсів однією з найбільш гнучких є React, з понад 15 мільйонами щотижневих завантажень [1]. React – це декларативна бібліотека на основі компонентів.

Деякі front-end фреймворки вирішують проблему управління даними «з коробки», але довгий час для таких програм React були потрібні сторонні рішення.

Наразі існує значна кількість бібліотек, які реалізують різні методи управління станом в односторінкових веб-застосунках, що розроблені за допомогою React. Ці бібліотеки активно розвиваються, та й нові рішення з’являються доволі часто. Через це з’являється проблема вибору доцільного методу для розробки нового веб-застосунку.

Огляди методів управління станом зазвичай торкаються лише питання розробницького досвіду, тоді як вплив тих чи інших методів на швидкодію додатку, що розробляється, не розглядається. Це є вагомим недоліком цих оглядів чи досліджень, адже швидкодія є важливим параметром, який впливає як на бізнес, так і на користувача.

React вже довгий час залишається однією з найпопулярніших бібліотек для створення веб-застосунків, а також позиціонується як один з найбільш оптимізованих з точки зору швидкодії метод. В той же час існує недостатньо інформації про вплив різних методів управління станом на веб-додатки, не розглядається важливість швидкодії, що призводить до неможливості визначити найбільш підходящий метод для різних проектів.

Таким чином обрана тема є актуальна як для виробництва, так і для розвитку науки, адже результати даної роботи дозволять приймати більш якісні рішення у виборі методу для управління станом та можуть бути гарною відправною точкою у подальших дослідженнях, пов’язаних як зі швидкодією веб-застосунків, так і з методами управління станом.

Метою роботи є визначення найбільш оптимізованого методу для управління станом в односторінкових веб-додатках з точки зору швидкодії.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– зробити аналіз та порівняння існуючих методів управління станом з точки зору об’єкта дослідження;

– визначити можливість впливу предмету дослідження на об’єкт та визначити метрики, за допомогою яких буде проведено оцінювання;

– розробити веб-застосунок для проведення експерименту;

– виміряти та підрахувати значення обраних метрик для кожного з методів управління станом;

– надати рекомендації щодо використання кожного з методів.

Об’єктом дослідження є швидкодія односторінкових веб-застосунків, які розроблені за допомогою бібліотеки React.

Предметом дослідження є методи для управління станом для веб-застосунків.

Методами дослідження є проведення вимірів розмірів вихідного коду та подальші обчислення метрик, які базуються на цих даних. Також у якості методу дослідження використано вимірювання швидкодії за допомогою спеціальної розробницької панелі приладів у браузері Chrome.

Отримані результати дослідження можуть бути використані в процесі прийняття рішення щодо методу управління станом для майбутній веб-застосунків. Також можливе подальше продовження дослідження в рамках інших методів управління станом чи на прикладі більш складних веб-застосунків.

Результати даної кваліфікаційної роботи було представлено на двох наукових конференціях:

– Дванадцята міжнародна науково-технічна конференція «Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління» [2];

– 6th International Conference on Computational Linguistics and Intelligent Systems (Scopus) May 12–13, 2022, Gliwice, Poland [3].

1 АНАЛІЗ ПРОБЛЕМНОЇ ГАЛУЗІ

1.1 Аналітичний огляд

Процес створення веб-додатків з роками еволюціонує і стає легшим. Працюючи з різними абстракціями високого рівня, розробники можуть проектувати та впроваджувати складні системи без необхідності постійно вирішувати одні й ті ж самі фундаментальні питання.

Цей швидкий розвиток має як переваги, так і недоліки. Очевидною перевагою є те, що щодня з’являється щось нове: нова бібліотека, новий підхід до тих чи інших речей, що робить повсякденну рутину веб-розробника досить різноманітною. Різні фреймворки та бібліотеки пропонують швидке рішення практично будь-якого завдання, вводячи в код програми різного ступеня обмеження.

Користувацькі інтерфейси створені для того, щоб користувачі могли зручно використовувати різноманітні системи, отримувати доступ до інформації та взаємодіяти з нею. Дуже поширена реалізація інтерфейсу як веб-застосунку, адже для користування таким додатком необхідно лише підключення до мережі Інтернет та браузер. Користувачі можуть використовувати будь-який прилад, який їм до вподоби: персональний комп’ютер, мобільний телефон чи навіть ігрову приставку або автомобіль.

Таке різноманіття призвело до того, що початкова архітектура веб-додатків стала втрачати свою актуальність, і почалися дослідження та спроби покращити наявні підходи та рішення для забезпечення швидкодії створюваних додатків.

Довгий час веб-додатки створювались за архітектурою багатосторінкових додатків. Багатосторінкові програми працюють «традиційним» способом. Будь-яка зміна в браузері (наприклад, відображення або передача даних) передбачає отримання нових сторінок із сервера [4]. У такій архітектурі уся логіка залишається на сервері, тоді як клієнт лише запитує та відображає сторінки. Однак з розвитком веб-технологій з’явилась можливість уникнути перемальовування цілої веб-сторінки у випадках, коли необхідно змінити лише невеликий обсяг даних.

Так почав розвиватися новий підхід до створення користувацьких інтерфейсів: реалізація веб-додатка як односторінкової програми, або SPA. Мета

SPA, яку можна вважати основною, це – запропонувати швидші переходи між користувачами та зробити веб-додаток більш схожим на рідну програму, яка працює локально на пристрої – у багатьох сенсах односторінкова програма і є локальною програмою, хоча її треба завантажувати з мережі Інтернет, а не з локального диска під час її запуску [5].

Створення односторінкових додатків підтримується багатьма фронт-енд фреймворками та бібліотеками. Також сучасні дослідження показують, що популярні фреймворки навіть більше підходять для створення односторінкових, ніж багатосторінкових додатків. Дослідження «Comparison of Front-End Frameworks for Web Applications Development» [6] підкреслило перехід до створення односторінкових додатків замість традиційного підходу до створення багатосторінкових програм. У цьому дослідженні були розглянуті три основні інтерфейсні фреймворки, Angular [7], Vue [8] та React, і виявилося, що всі вони підтримують створення односторінкових додатків краще або на тому ж рівні, що і багатосторінкових.

Найпопулярніші бібліотеки для створення користувацьких інтерфейсів за архітектурою односторінкових додатків регулярно стають об’єктами різноманітних досліджень. Вони розглядаються як окремо, так і в порівнянні один з одним. Оскільки ці методи побудови інтерфейсів постійно розвиваються, то й результати досліджень відмінні на різних версіях.

Наприклад, у порівнянні швидкодії односторінкового застосунку на прикладі React та Angular [9] останній мав кращі показники, тоді як висновки інших досліджень зовсім інші.

Так, у дослідженні чотирьох фреймворків з точки зору швидкодії взаємодії з об’єктною моделлю документа [10], React зайняв перше місце, обійшовши Vue, Angular та Svelte [11]. В результаті цього дослідження було отримано та порівняно багато показників: технічні показники, популярність та інше. React показав найкращий результат у цьому дослідженні і рекомендований як один з найкращих варіантів для розробки інтерфейсних додатків, оскільки він не має помітних недоліків і, в той же час, в нього є багато сильних сторін. Також React отримав

найвищий результат у двох технічних тестах і загалом мав показав дуже задовільні показники.

React, декларативна бібліотека на основі компонентів, являється однією з найбільш гнучких бібліотек для побудови користувацьких інтерфейсів серед найпопулярніших.

Гнучкість React, зокрема, полягає в тому, що він тривалий час позиціонується як бібліотека, яка вирішує лише завдання презентації, що дозволяє будувати інтерфейси користувача у вигляді дерева компонентів.

Серед переваг React можна виділити наступні:

– віртуальний DOM;
– широкий набір інструментів;
– масштабованість;
– та багато інших.

Створення простого додатка за допомогою React не може вважатися надто складним завданням, оскільки це можна зробити без будь-яких додаткових бібліотек. Ця бібліотека широко використовується для побудови користувацьких інтерфейсів як у маленьких, так і у великих веб-додатках, які потребують для свого створення також використання сторонніх бібліотек.

React обирають за рахунок того, що це швидкий та легкий інструмент, однак швидкодія веб-програми є складним значенням, яке не залежить лише від бібліотеки для створення інтерфейсу користувача. Великі веб-додатки працюють з величезною кількістю даних і мають структуру з багатьох компонентів, об’єднаних у сторінки зі структурами складних дерев, що ставить перед розробниками завдання підтримувати правильність усіх даних у програмі.

1.2 Виявлення проблем та актуалізація рішень

Деякі фреймворки для створення користувацького інтерфейсу вирішують проблему управління даними «з коробки», але довгий час для таких програм React були потрібні сторонні рішення. В результаті були створені різноманітні бібліотеки, які стали своєрідним стандартним вибором серед розробників.

Метою таких бібліотек є організація та керування потоком даних у зовнішніх програмах, у деяких випадках, незалежно від того, написана програма на React чи ні.

Дослідження методів управління станом у веб-додатках також має місце. Наприклад, у «Comparison of web application state management tools» [12] було розглянуто такі методи управління станом, як NgRx, Ngxs, Redux, Vuex. У дослідженні було розглянуто п’ять критеріїв:

– метрики коду;
– структура рішення;
– наявність готових реалізацій;
– підтримка спільноти та вимірювання швидкодії.

У цьому дослідженні також була проведена серія тестів, щоб проаналізувати роботу кожної бібліотеки.

Однак деякі з розглянутих бібліотек залежать від фреймворку і не можуть використовуватися з будь-якою бібліотекою для створення інтерфейсів користувача. Якщо розглядати React, то лише одна бібліотека серед досліджуваних може бути застосована до таких додатків.

Беручи до уваги те, що найчастіше при виборі інструментів для створення нового веб-додатку в першу чергу обирається бібліотека для створення користувацького інтерфейсу, а вже потім — метод управління станом у додатку та інші допоміжні бібліотеки, дослідження бібліотек, які можуть бути використані з певною бібліотекою створення інтерфейсу користувача, можна вважати більш доцільним.

Приклад такого дослідження – це «Comparison of State Management Solutions between Context API and Redux Hook in ReactJS» [13]. В даному дослідженні розглядаються та порівнюються вибіркові проекти з відкритим вихідним кодом. У якості методів управління станом було обрано Redux та Context API. Ці два підходи були розглянуті з точки зору зручності використання, а також було обчислено метрику складності вихідного коду. Окрім цього в даному дослідженні було розглянуто використання пам’яті на різних девайсах, що доцільно для розуміння впливу цих підходів на швидкодію проектів.

У цілому в дослідженні було розглянуто наступні критерії:

– впровадження;
– відстеження змін;
– встановлення додаткових пакетів;
– складність кодової бази;
– споживання ресурсів;
– швидкість обробки та масштабованість.

В результаті цього дослідження було зроблено висновок, що Redux є найбільш підходящим рішенням для великих проектів, тоді як Context API можна використовувати лише у невеликих проектах.

Розглянуті в дослідженні критерії та методи не дають повної картини поточного стану методів для управління станом для односторінкових додатків, що розробляються з використанням React.

Враховуючи, що обсяг даних у складних зовнішніх програмах може бути величезним, використання неправильного підходу до керування станом може призвести до серйозних проблем із продуктивністю, особливо на слабких пристроях, таких як смартфони, планшети та старі комп’ютери.

1.3 Постановка задачі

Доцільний метод управління станом потрібен майже у будь-якому односторінковому веб-додатку. Якщо додаток розробляється за допомогою бібліотеки для створення користувацьких інтерфейсів React, то на вибір розробників пропонується велике різноманіття бібліотек, які організують цей процес різноманітними засобами.

Одним з недоліків такої наявності різноманітних інструментів для вирішення однієї і тієї ж проблеми є той факт, що немає реальних стандартів, які можна використовувати при виборі технологій для розробки різних додатків. Це може призвести до того, що обрані технології не дуже доцільні для використання з точки зору продуктивності [14], а просто найбільш поширені або з якими розробник, який приймає рішення, вже має позитивний досвід.

Веб-сайти з високою швидкодією покращують користувацький досвід. Пришвидшуючи веб-застосунок можна покращити користувацький досвід за рахунок прискорення доставки вмісту. Користувачі мають схильність до того, щоб надавати перевагу веб-застосункам, які працюють та завантажуються швидше. До того ж, швидкодія веб-сайтів впливає не тільки на користувачів, а й на позицію веб-сайту в результатах пошуку Google [15].

Це робить дослідження методів управління станом з точки зору швидкодії справді важливим. Метою цього дослідження є порівняння методів Redux, Recoil та React Hooks для керування станом у односторінкових програмах React та надання майбутніх рекомендацій на основі отриманих результатів.

Виходячи з усього перерахованого вище, в рамках дослідження необхідно вирішити наступні завдання:

– розглянути існуючі методи управління станом в односторінкових веб-застосунках, які створені за допомогою React;

– проаналізувати та обрати методи управління станом для проведення дослідження;

– проаналізувати існуючі підходи до вимірювання швидкодії веб-застосунків;

– проаналізувати можливість впливу методів для управління станом на швидкодію веб-застосунків;

– визначити метрики, які будуть використані для проведення експерименту та подальшого оцінювання;

– визначити функціональні вимоги для веб-застосунку, який буде використано для проведення експерименту, та розробити його;

– виміряти та підрахувати значення обраних метрик для кожного з методів управління станом;

– порівняти та проаналізувати отримані дані;

– надати рекомендації щодо використання кожного з методів.

2 ВИБІР МЕТОДІВ УПРАВЛІННЯ СТАНОМ

2.1 Аналіз існуючих методів управління станом

Тисячі різних бібліотек реалізують різні методи управління станом веб-додатків: деякі з них підходять лише для тих чи інших бібліотек для створення користувацького інтерфейсу, інші ж більш універсальні в цьому питанні. Але таке різноманіття ускладнює вибір правильного підходу для конкретного проекту.

Окрім безпосередньо менеджеру пакетів для веб-додатків, ознайомитися з існуючими варіантами бібліотек для управління станом можна у різноманітних оглядових статтях у мережі Інтернет. Такі статті з’являються майже кожен день, та, найчастіше, не несуть в собі будь-якої корисної інформації, окрім власної думки автора щодо того, якою бібліотекою йому було зручніше користуватися.

Також кожного року проводиться опитування на веб-ресурсі State of JS [16]. Збираючи дані від тисяч розробників, щорічне дослідження стану JavaScript здатне визначити поточні та майбутні тенденції в екосистемі. Розробники проходять опитування, щоб допомогти дізнатися, які бібліотеки вони хочуть вивчати наступного року, які мають найкращі оцінки задоволеності та багато іншого.

Однак у поточному році дослідження шару даних не проводилось, але цим ресурсом можна скористатися для того, щоб скласти уявлення про трендові технології у веб-розробці.

2.1.1 Огляд методу управління станом Redux

У якийсь момент бібліотека для управління станом Redux стала дуже популярною та використовувалась майже у кожному новому веб-додатку. Ця бібліотека має свої переваги, такі як масштабованість, і допомагає зробити підтримку стану програми передбачуваною [17]. Але у Redux є і багато недоліків, які привели до подальшого розвитку менеджерів стану для веб-додатків.

Основна концепція Redux полягає в трьох принципах. По-перше, весь стан програми зберігається в одному центральному місці та має вигляд простого об’єкту. Цей об’єкт називається глобальним станом, чи store. По-друге, цей глобальний стан доступний тільки для читання – задля того, щоб внести там зміни, необхідно ініціювати відповідну дію. По-третє, зміни в глобальному стану

відбуваються за допомогою чистих функцій (це означає, що в цих функціях не повинно бути жодних побічних ефектів).

Кожен компонент веб-застосунку може мати прямий доступ до цього глобального стану без необхідності передавати властивості до дочірніх компонентів або використовувати функції зворотного виклику для надсилання даних до батьківського компоненту [18].

Потік даних у Redux є односпрямованим, що забезпечує його передбачуваність. Схему потоку показано на рисунку 1.

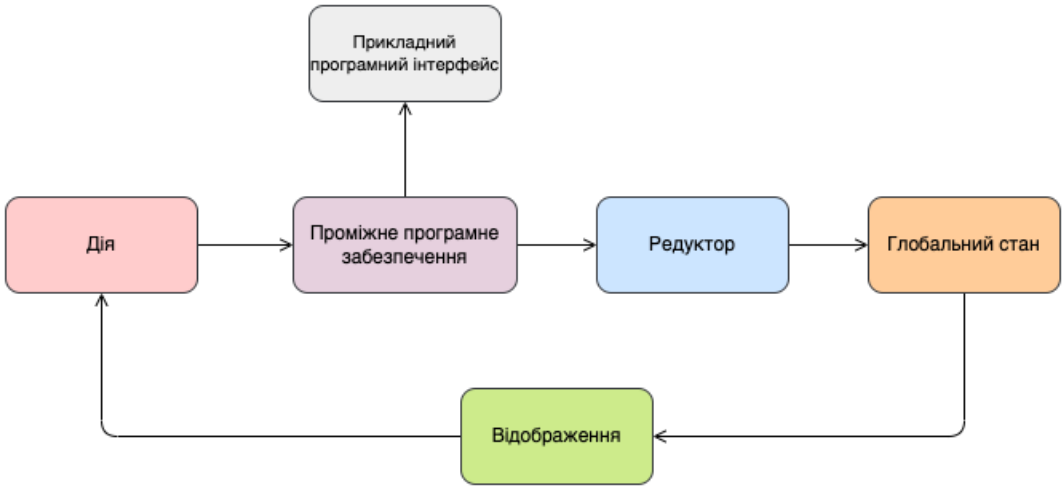

Рисунок 1 — Потік даних Redux

Побудувати архітектуру для складних додатків за допомогою Redux не є надто складним завданням, оскільки основні концепції цього вже розділяють різні обов'язки. Хоча кількість шаблонів, введених Redux, можна назвати занадто надмірним.

З Redux ми оперуємо такими поняттями, як дії та редуктори. За замовчуванням дії є синхронними і не підходять для таких завдань, як виклики прикладних програмних інтерфейсів, тому для цього необхідно встановлювати додаткове проміжне програмне забезпечення.

Дії в Redux – це звичайні об'єкти, які зазвичай мають наступну структуру:

```
{
  type: string;
```

```
  payload: any;
}
```

Тип дії дозволяє в подальшому диференціювати різні дії та стає у нагоді при логуванні.

Корисне навантаження може бути представлене будь-якою структурою даних та буде оброблено в спеціальному редукторі. Однак при використанні спеціальних бібліотек, які підключаються як проміжне програмне забезпечення, дії можуть перетворюються в асинхронні функції.

Редуктор – це чиста функція, яка приймає поточний стан і відправлену дію, потім обробляє його та повертає наступний стан.

Усі редуктори, а також додаткове проміжне програмне забезпечення об’єднуються в сховище, яке передається до програми React спеціальним постачальником компонентів.

Redux набув популярності за рахунок того, що структура, яку він запроваджував, могла використовуватися майже в будь-якому проекті, що полегшувало як задачу управління станом, так і адаптацію нових розробників, які вже мали досвід з цією бібліотекою.

Redux присутній у даних з опитування у п’яти роках: з 2016 по 2020 (у 2021 не проводилось відповідне опитування). На рисунку 2 можна побачити як змінювалось ставлення розробників, котрі брали участь в опитуванні [15], до цієї бібліотеки управління станом у веб-додатках.

У 2016 році частка тих, хто використовував Redux та бажав використовувати цю бібліотеку у наступних проектах була лише 33.5% та протягом трьох років збільшувалась, складаючи 47.7% у 2019 році. Однак, у 2020 році цей показник суттєво зменшився і лише 44.6% розробників відповіли, що будуть використовувати Redux як бібліотеку для управління станом у подальших проектах.

Також значно знижувався відсоток тих, хто цією бібліотекою не користувався, але був у ній зацікавлений. Це може свідчити як про те, що ці розробники мали нагоду спробувати цю бібліотеку у проектах, так і про те, що їхній

інтерес просто зник. Про останнє також свідчить те, що не зацікавлених у цій бібліотеці в 2016 було лише 9.6% респондентів, тоді як в 2020 «так» відповіли вже 13.6% розробників.

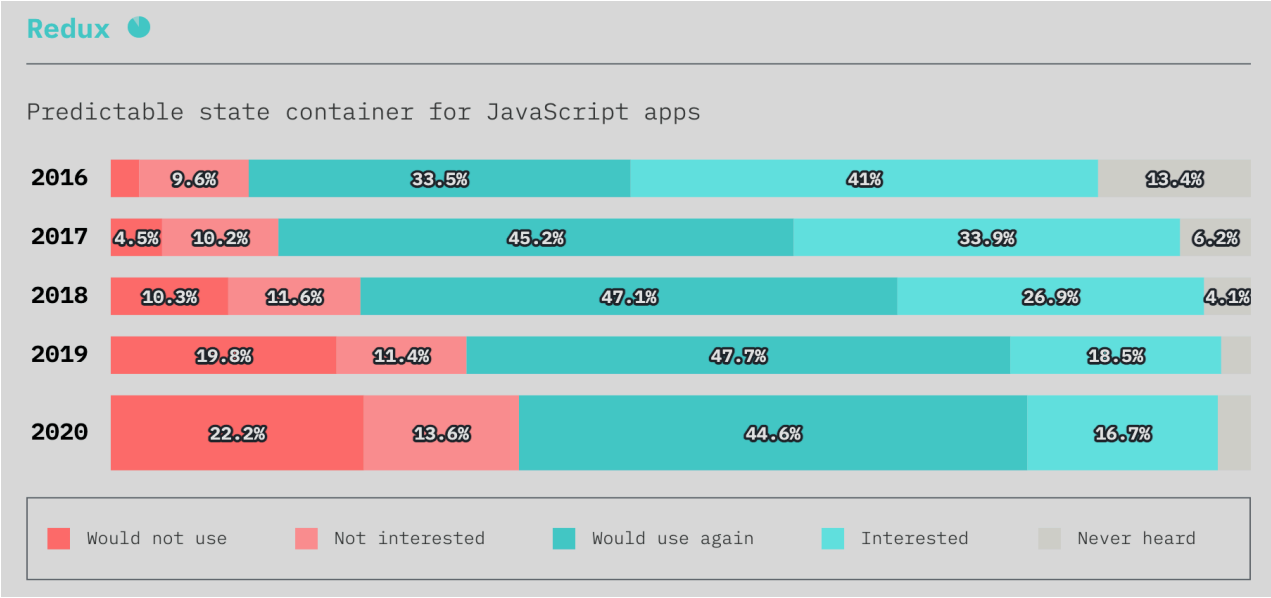

Рисунок 2 — Огляд думок щодо Redux за 2016-2022 роки

Важливим показником можна вважати кількість розробників, які відповіли, що не будуть використовувати Redux у наступних проектах. Цей показник в 2020 році склав 22.2% – майже половина від кількості розробників, що бажають використовувати Redux і надалі.

Загалом складається негативна картина щодо задоволення розробниками цим методом. Але за даними того ж дослідження, Redux залишається найпопулярнішою бібліотекою, яку використовує або використовувала найбільша кількість респондентів.

2.1.2 Огляд методу управління станом React Hooks

Як вже було зазначено, довгий час в бібліотеці React не було реалізовано рішень для управління станом веб-додатків, через що розробники шукали якість сторонні методи та бібліотеки. Але в React v16.8 були випущені Hooks.

Hooks вирішують широкий спектр, здавалося б, не пов’язаних проблем у React і дозволяють повторно використовувати логіку стану без зміни ієрархії

компонентів [19]. Це призвело до ще більшої плутанини при виборі підходу до управління станом додатків.

Для реалізації React Hooks не потрібні інші залежності, крім самої бібліотеки React з версією від 16.8.

Підхід React Hooks дає можливість реалізувати всю логіку безпосередньо всередині компонентів, будь то базова операція зі станом, взаємодія з API браузера або мережевий запит.

Хук useState, який є частиною функціоналу, що надається бібліотекою React, надає можливість використовувати стан у функціональному компоненті. Раніше подібна можливість була лише у тих компонентів, що були створені за допомогою класів.

Хук useEffect дозволяє виконувати побічні ефекти в компонентах і нагадує методи життєвого циклу в класах. Цей хук приймає функцію, що містить імперативний код, у якому можуть бути різні ефекти, тобто цей код може впливати на змінні, що знаходяться зовні.

Зміни, підписки, таймери, логування та інші побічні ефекти не дозволяються всередині основного тіла функціонального компонента (яке називається етап рендеру). Це призводить до заплутаних помилок та невідповідностям у користувацькому інтерфейсі.

Функція, передана в useEffect, буде запущена після того, як вивід рендеру з’явиться на екрані.

Хук useCallback повертає мемоізовану функцію зворотнього виклику , котра змінюється лише тоді, коли одна з її залежностей змінюється.

Мемоізація – це метод оптимізації, який в основному використовується для прискорення комп'ютерних програм шляхом зберігання результатів дорогих викликів функцій та повернення кешованого результату, коли виклики на однакових вхідних даних відбуваються знову [20].

Це корисно при передачі функцій зворотнього виклику до оптимізованих дочірніх компонентів, що покладаються на рівність посилань задля уникнення непотрібних рендерів.

Хук useMemo повертає мемоізоване значення. Цей хук повторно обчислить мемоізоване значення лише при зміні однієї з залежностей. Така оптимізація допомагає уникнути вартісних обчислень при кожному рендері.

Хук useRef поверне змінний об’єкт рефу, властивість .current якого ініціалізується переданим аргументом (initialValue). Повернутий об’єкт буде зберігатись протягом всього часу життя компонента.

Поширеним випадком використання є імперативний доступ до потомків. Проте useRef() є корисним не тільки для простого атрибута ref. Він згодиться для постійного збереження будь-якого змінного значення подібно до використання полей екземпляра класу.

Цей простий підхід у масштабі створює багато дубльованого коду, а також обмежує логіку певними компонентами, що призводить до тісно пов’язаного коду. Це збільшує складність підтримки коду та знижує ступінь можливості повторного використання коду.

Хоча цього можна уникнути, створюючи та використовуючи власні хуки. Власний хук – це функція, яка використовує вбудовані React Hooks для забезпечення більш складної функціональності. Таким чином може бути досягнута інкапсуляція всієї логіки, пов'язаної з API або іншими маніпуляціями зі станом веб-додатку.

2.1.3 Огляд методу управління станом Recoil

Recoil – це експериментальний набір утиліт для управління станом за допомогою React [21]. Recoil розробляється компанією Facebook, як і безпосередньо React. Також незважаючи на те, що ця бібліотека ще знаходиться у стані активної доробки, вона займає перше місце серед інструментів для управління станом у веб-додатків, що розробляються за допомогою React, у підпорядкованому списку кращих технологій, що відносяться до веб-розробки [22].

Recoil визначає граф напрямку ортогональним, але також внутрішнім і приєднаним до дерева React. Зміни стану відбуваються від коренів цього графа (які називаються атомами) через чисті функції (які називаються селекторами) до компонентів.

Такий підхід забезпечує наступні переваги та можливості:

– прикладний програмний інтерфейс без шаблонів, де спільний стан має той самий простий інтерфейс отримання/налаштування, що й локальний стан React (але може бути інкапсульований за допомогою редукторів тощо, якщо потрібно);

– можливість сумісності з паралельним режимом та іншими новими функціями React, коли вони стануть доступними;

– визначення стану є інкрементальним і розподіленим, що робить можливим поділ коду;

– стан можна замінити похідними даними без зміни компонентів, які його використовують;

– похідні дані можуть переміщатися між синхронними та асинхронними без зміни компонентів, які їх використовують;

– навігацію можна розглядати як першокласну концепцію, навіть закодувати переходи станів у посилання.

Як вже було зазначено, атоми та селектори є основними концепціями, якими оперує Recoil.

Атоми є одиницями стану. Вони можуть оновлюватися та на них можна підписатися: коли атом оновлюється, кожен компонент, який на нього підписаний, повторно відтворюється з новим значенням. Атоми також можна створювати під час виконання. Атоми можна використовувати замість стану локального компонента React. Якщо той самий атом використовується з кількох компонентів, усі ці компоненти мають однаковий стан.

Для створення атому необхідно використати відповідну функцію:

```
const exampleState = atom({
  key: 'exampleState',
  default: 24,
});
```

Атомам потрібен унікальний ключ, який використовується для налагодження, збереження та для певних розширених прикладних програмних

інтерфейсів, які дозволяють бачити карту всіх атомів. Як і стан компонента React, вони також мають значення за замовчуванням.

Загалом структура атому схожа на поєднання структури дії з Redux та звичайного стану компоненту.

Щоб прочитати й записати в атом із компонента, використовується хук під назвою useRecoilState. Це так само як useState у React, але тепер цей стан можна поділити між компонентами:

Селектор – це чиста функція, яка приймає атоми або інші селектори як вхідні дані. Коли ці вищестоящі атоми або селектори оновлюються, функція селектора буде повторно викликана. Компоненти можуть підписатися на селектори так само, як і атоми, і 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 мінімального стану. Оскільки селектори відстежують, які компоненти вони потребують і від якого стану вони залежать, вони роблять цей функціональний підхід дуже ефективним.

З точки зору компонентів, селектори та атоми мають однаковий інтерфейс і тому можуть бути замінені один одним.

2.2 Визначення методики проведення дослідження

Вплив обраного підходу до управління станом можна спостерігати як на вихідний код (у випадку інтерфейсної програми це розмір файлу JavaScript), так і на вимірювання безпосередньо запущеної програми.

Важливо думати про продуктивність на всіх етапах розробки програми, а не тільки тоді, коли настає критичний момент. Можна сказати, що оцінка програмного забезпечення на стадії проектування має високе практичне застосування, оскільки дозволяє провести оцінку програмного забезпечення до його початку, що, в свою чергу, дозволяє врахувати більшість можливих ризиків проекту та етапів розробки. У свою чергу, це дозволяє порівняти, які витрати необхідні і яка майбутня вартість проекту [23].

Швидкодію веб-додатка можна дуже легко погіршити, якщо використовувати будь-які неправильні бібліотеки, не тільки для управління станом. А метрики та їх граничні значення зазвичай визначаються індивідуально для кожного проекту, якщо такі є.

Google створив модель під назвою WebVitals для надання розробникам рекомендацій щодо розробки в процесі проектування та створення веб-додатків, які надзвичайно важливі для забезпечення належного користувацького досвіду в Інтернеті.

WebVitals – це корисний інструмент для оцінки продуктивності веб-додатків.

Core Web Vitals – це частина метрик Web Vitals, які використовуються для оцінки веб-сторінок і включені до всіх інструментів Google. Кожен показник Core Web Vitals є окремим аспектом досвіду взаємодії користувача з сайтом, що вимірюється в польових умовах і відображає реальні дії з досягнення критично важливого результату, орієнтованого на користувача.

Core Web Vitals містить такі показники:

– відображення найбільшого вмісту;
– затримка першого введення;
– сукупний зсув макета.

Показники, визначені таким чином, є емпіричним виміром реального досвіду користувача, який аналізує Google і які базуються на ключових потребах користувачів: швидкість завантаження сторінки, інтерактивність програми, стабільність, досвід роботи з вмістом тощо [24].

Крім показників Core Web Vitals, існують також інші показники для різних етапів взаємодії між користувачем і сторінкою. Хоча на більшість із цих показників впливає не підхід до управління станом, а інші частини користувацього веб-додатка. Тому було вирішено не використовувати WebVitals безпосередньо.

Тим не менш, метрики, створені Google, були враховані. Передбачалося, що найбільший вплив вибору менеджера стану можна спостерігати на розмірі збірки та під час взаємодії користувача зі сторінкою.

Для проведення дослідження та порівняння обраних методів управління станом були визначені наступні показники:

– дельта відсотка в кількості коду, необхідного для впровадження базових методів для сутностей в проекті;

– відсоток повторно використаного коду між двома сутностями в проекті;

– передбачуваний подальший темп зростання розміру коду для нових сутностей;

– дельта в розмірі коду, необхідного для реалізації керування станом, у кожному з методів, що розглядаються, у збірці, яка призначена для розгортання у робочому середовищі;

– дельта пам'яті, що використовується для зберігання однакового обсягу даних у веб-застосунку;

– дельта часу для операції оновлення стану.

Чим більший розмір збірки, тим більше потрібно пристрою користувача, щоб завантажити всі файли програми та запустити його. Час, який потребується для завантаження сторінки та її інтерактивності, є дуже важливим показником, оскільки він суттєво впливає на потенційного користувача.

Логіка взаємодії з даними зазвичай розділяється (це рекомендується як частина принципів SOLID [25], а саме принципом єдиної відповідальності). Критерії, за якими відбувається поділ логіки та формування архітектури кожного проекту, можуть бути будь-якими. Наприклад, одними з найпопулярнішими підходами є розділення по типам та функціям, але є й інші так звані «гібридні» варіанти [26].

Для зручності будемо іменувати одну одиницю з архітектури логіки сутністю.

Що стосується показників, пов’язаних із взаємодією з користувачами, то доводити їх важливість немає потреби, оскільки вони у своїй основі є ядром користувацького досвіду. І, крім того, неправильно обраний метод для керування станом у веб-застосунку може призвести до плутанини в потоці даних та втрат бізнесу.

3 СТВОРЕННЯ ПРОГРАМНОЇ СИСТЕМИ ДЛЯ ДОСЛІДЖЕННЯ

3.1 Функціональні вимоги

Щоб вивчити вплив різних підходів для управління станом на швидкодію веб-застосунку, було розроблено тестовий односторінковий веб-застосунок з використанням React.

До розробленого веб-застосунку було визначено наступні функціональні вимоги:

– програма відображає список дописів, що отримуються із стороннього прикладного програмного інтерфейсу;

– кожен допис має власну назву, зміст та пов'язаного з ним автора (користувача);

– кожен допис можна відредагувати та видаліти;

– новий допис може бути створено;

– додаткові дані про автора допису отримуються за допомогою надсилання ідентифікатора користувача до стороннього прикладного програмного інтерфейсу;

– дані автора допису можуть бути оновлені;

– можна створити нового користувача;

– можна видалити вже існуючого користувача.

Окрім вищезазначеного, усі дані про дописи мають зберігатися у спеціальному сховищі всередині веб-додатку, а дані про кожного з авторів слід отримувати окремо для кожної публікації.

Згідно з перерахованими функціональними вимогами, у тестовому застосунку має бути організовано роботу з даними, що відносяться до двох сутностей: користувач та допис.

Таким чином, реалізація одних і тих же функціональних вимог за допомогою різних методів для управління станом дає змогу провести їхнє порівняння без будь-яких припущень чи модифікації даних результату, оскільки експеримент буде проводитися на одних і тих же даних.

Відповідно до функціональних вимог, зазначених вище, для сутності допису мають бути реалізовані такі методи взаємодії з прикладним програмним інтерфейсом:

– отримати список дописів;
– створити новий допис;
– оновити існуючий допис;
– видалити існуючий допис.

І, відповідно, для сутності користувача необхідно реалізувати наступні методи:

– отримати користувача за унікальним ідентифікатором;
– створити нового користувача;
– оновити існуючого користувача;
– видалити існуючого користувача.

Презентаційна частина програми має бути розроблена за допомогою бібліотеки React та має залишатися незмінною під час дослідження, тоді як управління станом буде реалізується окремо за допомогою обраних.

3.2 Розробка презентаційної частини

Для створення користувацького інтерфейсу було використано бібліотеку React. Ця бібліотека передбачає, що інтерфейс формується як композиція із різних компонентів.

Для того, щоб правильно сформувати набір компонентів, які необхідні для реалізації користувацького інтерфейсу, було створено макети за допомогою сервісу Moqups [27]. Це спрощений веб-додаток, який допомагає створювати каркаси, макети, діаграми та прототипи та співпрацювати в реальному часі.

Основний екран веб-застосунку передбачає відображення списку дописів. Кожен допис має назву, текст та автора, що відображено на прикладі розробленого макету зображено на рисунку 3.

Рисунок 3 – Макет основного екрану

Після створення макетів стало можливо провести виділення компонентів. Таким чином, наступні компоненти були визначені як необхідні для реалізації:

– App;
– PostView;
– PostList;
– Post;
– PostForm;
– UserView;
– User;
– UserForm.

Розмітка компонентів описується за допомогою JSX. Це розширення синтаксису для JavaScript. JSX може нагадувати мову шаблонів, але з усіма перевагами JavaScript. Замість того, щоб штучно відокремити технології, розмістивши розмітку і логіку в окремих файлах, React розділяє відповідальність між вільно зв’язаними одиницями, що містять обидві технології і називаються “компонентами”.

Для придання компонентам більш приємного вигляду було використано CSS – спеціальну мову стилю сторінок, що використовується для опису їхнього зовнішнього вигляду. CSS є основною технологією всесвітньої павутини, поряд із HTML та JavaScript [28].

За замовчуванням CSS, імпортований з JavaScript, є глобальним. Якщо два файли CSS визначають однакові назви класів, вони потенційно можуть зіткнутися та перезаписати один одного. Щоб вирішити цю проблему, були винайдені модулі CSS.

Модулі CSS розглядають класи, визначені в кожному файлі, як унікальні. Кожне ім’я класу перейменовується, щоб включати унікальний хеш, а словник експортується в JavaScript, щоб дозволити посилатися на ці перейменовані імена класів.

Після реалізації кожного з компонентів вони були скомпоновані та об’єднані в компонентне дерево, яке зображено на рисунку 4.

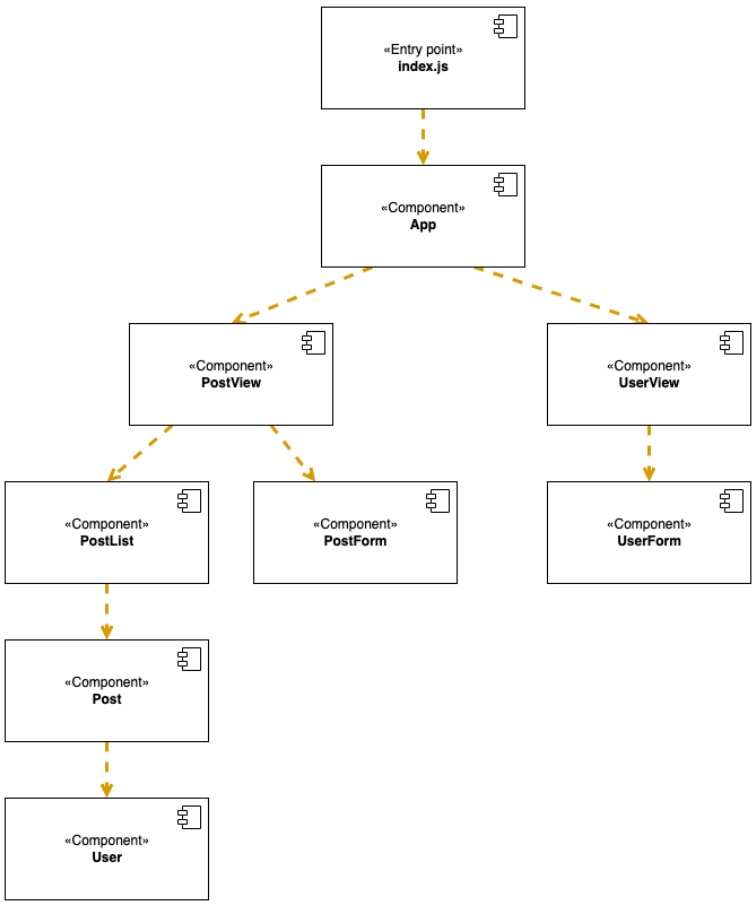

Рисунок 4 – Діаграма компонентів

Для того, щоб зібрати проект, було використано спеціальний інструмент – Parcel [29]. Parcel поєднує в собі чудовий досвід розробки з коробки з масштабованою архітектурою, яка може перенести проект від початку до масового виробничого застосування.

Цей інструмент надає можливість використовувати різноманітні корисні функції, такі як сервер розробки, гаряче перезавантаження, та багато інших, без необхідності створювати величезні файли з конфігурацією.

На рисунку 5 зображено знімок екрана з розробленим веб-застосунком.

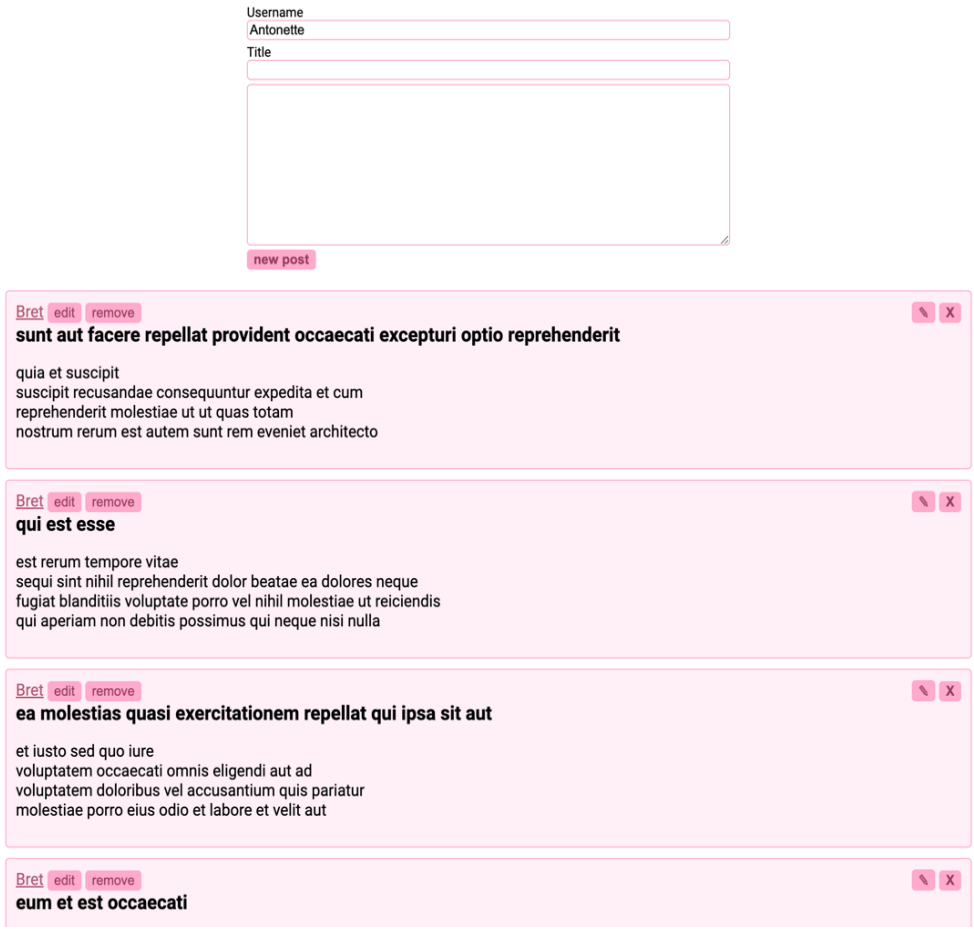

Рисунок 5 – Знімок екрана з розробленим веб-застосунком

У застосунку було реалізовано всі функціональні вимоги, необхідні для коректного проведення дослідження.

3.3 Реалізація управління станом

3.3.1 Реалізація управління станом за допомогою Redux

Для реалізації управління станом за допомогою Redux було встановлено наступні пакети:

– redux;

– react-redux;

– redux-thunk.

Використання Redux передбачає створення усіх структур, що впроваджує даний підхід. Оскільки в розробленому застосунку існує дві сутності: допис та користувач, то наступні структури були створені для кожної з цих сутностей окремо:

– actions (дії);

– reducer (редуктор).

Також було створено файл, у якому відбувається ініціалізація загального стану застосунку та допоміжні методи для взаємодії з прикладним програмним інтерфейсом.

Приклад коду редуктора, що використовується для сутності допису, наведено у додатку Б.

Для того, щоб застосунок зміг отримувати дані з загального стану Redux було використано допоміжну функцію Provider, яку було імпортовано з пакету react-redux. У цю функцію було загорнуто кореневий компонент застосунку:

```
ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('root')
);
```

Для того, щоб отримати дані з загального стану, було використано метод connect, який надається пакетом react-redux. Нижче наведено приклад коду, необхідного для отримання списку дописів з глобального стану, а також функції, яка дозволяє ініціювати дію створення нового допису:

```
const mapStateToProps = state => ({
  posts: state.posts,
});

const mapDispatchToProps = dispatch => ({
  createPost: (data) => dispatch(postActions.create(data))
});

connect(mapStateToProps, mapDispatchToProps)(App);
```

Таким чином відбувається «підключення» компоненту до глобального стану. Рекомендованим підходом є поділ компонентів на компоненти-контейнери та презентаційні компоненти. Інакшими словами, частина компонентів у додатку має отримувати дані від батьківського компоненту та відображати їх, тоді як інші компоненти мають підключатися до глобального стану та отримувати дані чи можливість змінювати цей стан.

У розробленому додатку наступні компоненти мають роль компонентів-контейнерів та підключені до глобального стану:

– PostView;
– UserView;
– User.

Усі інші компоненти реалізовані як прості презентаційні компоненти та лише відображають дані, що передаються до них у якості параметрів.

3.3.2 Реалізація управління станом за допомогою React Hooks

Для реалізації управління станом додатку за допомогою React Hooks були використані наступні з базових хуків:

– useState;
– useMemo;
– useCallback;
– useEffect.

Для взаємодії з прикладним програмним інтерфейсом було створено власний хук під назвою useAPI. Код створеного хука наведено у додатку В.

Цей хук було імпортовано та використано у компонентах, яким необхідно взаємодіяти з прикладним програмним інтерфейсом, а саме:

– PostView;
– Post;
– UserForm.

У компоненті PostView відбувається запит до прикладного програмного інтерфейсу на отримання списку усіх дописів. Цей запит відбувається безпосередньо у хуку useEffect задля того, щоб ця взаємодія відбувалася лише один раз, а не на кожне відмальовування компоненту.

Нижче наведено частину коду компоненту PostView, що виконує описані маніпуляції.

```
const [posts, setPosts] = useState([]);

const { getAll } = useAPI('posts');

useEffect(() => {
  getAll(setPosts);
}, [setPosts]);
```

У якості залежності цьому хуку передається функція setPosts, адже вона необхідна для збереження списку постів у стані компонента, який створено за допомогою хуку useState.

Після того, як список дописів отримано з прикладного програмного інтерфейсу, він передається до дочірнього компоненту PostList у якості параметра.

Для створення нового допису окрім розробленого власного хуку також використовується хук useCallback, який дозволяє створювати мемоізовану функцію.

```
  const createPost = useCallback((postData) =>  {
    const callback = (newItem) => {
      setPosts(prev => [{...postData, id: Math.random() }, ...prev]);
    }
    create(postData, callback);
  }, [create, setPosts]);
```

Усі інші функції, які необхідні для взаємодії з прикладним програмним інтерфейсом, були створені за прикладом вищенаведених.

3.3.3 Реалізація управління станом за допомогою Recoil

Реалізація управління станом за допомогою бібліотеки Recoil в цілому схожа як з Redux, так і з React Hooks. Так, щоб підключити додаток, написаний на React до цієї бібліотеки, необхідно імпортувати компонент RecoilRoot та загорнути в нього кореневий компонент додатка. Це дуже схоже на використання компоненту Provider з бібліотеки Redux:

```
ReactDOM.render(
  <RecoilRoot>
    <App />
  </RecoilRoot>,
  document.getElementById('root')
);
```

Відмінність полягає у тому, що для Recoil нема необхідності створювати єдиний об’єкт глобального стану та передавати його напряму.

Для збереження списку дописів було створено спеціальну одиницю стану додатку – атом. Для цього було імпортовано спеціальну функцію, яку надає ця бібліотека для управління станом.

```
import { atom } from "recoil"

export const postListState = atom({
  key: "postListState",
  default: [],
});
```

Для отримання даних з атому були використані наступні методи: useSetRecoilState та useRecoilValue. Останній дозволяє лише читання стану, отже у випадках, коли була необхідність у модифікуванні значення, що зберігається в атому, було використано саме метод useSetRecoilState.

```
const [posts, setPosts] = useRecoilState(postListState);
```

Для цього у метод useSetRecoilState було передано посилання на атом зі станом дописів, код створення якого наведено вище.

Також для отримання даних за допомогою Recoil були створені селектори. Нижче наведено приклад коду, який виконується для отримання інформації про користувача, що є автором певного допису:

```
const userQuery = selectorFamily({
  key: 'User',
  get: userID => async () => {
    const response = await getUserInfo({userID});
    if (response.error) {
      throw response.error;
    }
    return ({
      username: response.username,
      avatar: response.avatar,
    })
  },
});
```

Як можна побачити з наведеного коду, безпосередньо взаємодія з прикладним програмним інтерфейсом відбувається за допомогою окремого сервісу, тоді як використання селектору дозволяє реалізувати шаблон проектування MVVM (Модель, Вигляд, Модель вигляду) додержуючись принципу єдиної відповідальності та на змішуючи увесь код, що відноситься до взаємодії з даними.

4 ОПИС ПРОВЕДЕНИХ ДОСЛІДЖЕНЬ

У межах даної роботи було проведено два типи вимірювань:

– вимірювання розмірів коду;

– вимірювання швидкодії.

Для вимірювання розмірів коду проекту було використано онлайн-інструмент bytesizematters [30].

Результати, що були отримані при вимірюванні коду, наведено в таблиці 1.

Таблиця 1 – Вимірювання розмірів вихідного коду

№	Метрика	Вимірювання версії з Hooks	Вимірювання версії з Redux	Вимірювання версії з Recoil
1	Методи першої сутності	2.01 KB	3.79 KB	2.47 KB
2	Методи другої сутності	742 B	2.34 KB	826 B
3	Спільний код сутностей	1.02 KB	1.44 KB	1.48 KB
4	Загальний код для обох сутностей	2.75 KB	6.19 KB	3.29 KB
5	Розмір коду для управління станом у production версії	1.26 KB	16.49 KB	59.88 KB

Наступна формула (1) використовується для розрахунку відсотка дельти в кількості коду, необхідного для впровадження базових методів для однієї сутності як у новому проекті, так і в проекті з реалізованими іншими сутностями:

$$V_{\Delta bm} = \frac{V_1 - V_2}{V_1} * 100\%, \quad (1)$$

де V_1 – це розмір коду, який був розроблений для реалізації методів першої сутності в КБ,

V_2 – це розмір коду для другої сутності відповідно.

Результати розрахунків представлені в таблиці 2.

Таблиця 2 – Дельта коду базових методів

Метрика	Вимірювання версії з Hooks	Вимірювання версії з Redux	Вимірювання версії з Recoil
$V_{\Delta bm}$	63.95%	32.26%	67.34%

Важливість кількості коду, який можна повторно використовувати, зростає пропорційно кількості сутностей, яких може бути чимало у великих проектах. Отже, його варто обчислити. Для обчислення відсотка повторно використаного (спільного) коду використовується така формула:

$$V_r = \frac{V_1}{V_2} * 100\%, \tag{2}$$

де V_1 – це розмір коду, який використовується обома сутностями в КБ,

V_2 – загальний розмір коду для обох сутностей у КБ.

Результати розрахунків представлені в таблиці 3.

Таблиця 3 – Відсоток повторно використаного коду для обох сутностей

Метрика	Вимірювання версії з Hooks	Вимірювання версії з Redux	Вимірювання версії з Recoil
V_r	37.09%	23.26%	44.99%

Припускаючи, що наступні об’єкти потребуватимуть такий самий обсяг коду для реалізації, що й другий об’єкт, швидкість зростання можна розрахувати за такою формулою:

$$V_g = \frac{V_1 - V_2}{V_2} * 100\%, \tag{3}$$

де V_1 – загальний розмір коду для обох сутностей у КБ,

V_2 – розмір коду для першої сутності.

Результати розрахунків представлені в таблиці 4.

Таблиця 4 – Темпи зростання коду нових сутностей

Метрика	Вимірювання версії з Hooks	Вимірювання версії з Redux	Вимірювання версії з Recoil
V_g	36.82%	63.32%	33.20%

Для вимірювання продуктивності запущеної веб-програми можна використовувати Chrome DevTools. Chrome DevTools – це інтерактивна інформаційна панель, яка містить багато різних інструментів для веб-розробників, і вона вбудована у браузер Chrome.

Серед представлених вкладок одна під назвою Performance може надати багато важливої інформації та пролити світло на те, як вона насправді працює під капотом. Приклад того, як може виглядати ця вкладка, показано на рисунку 6.

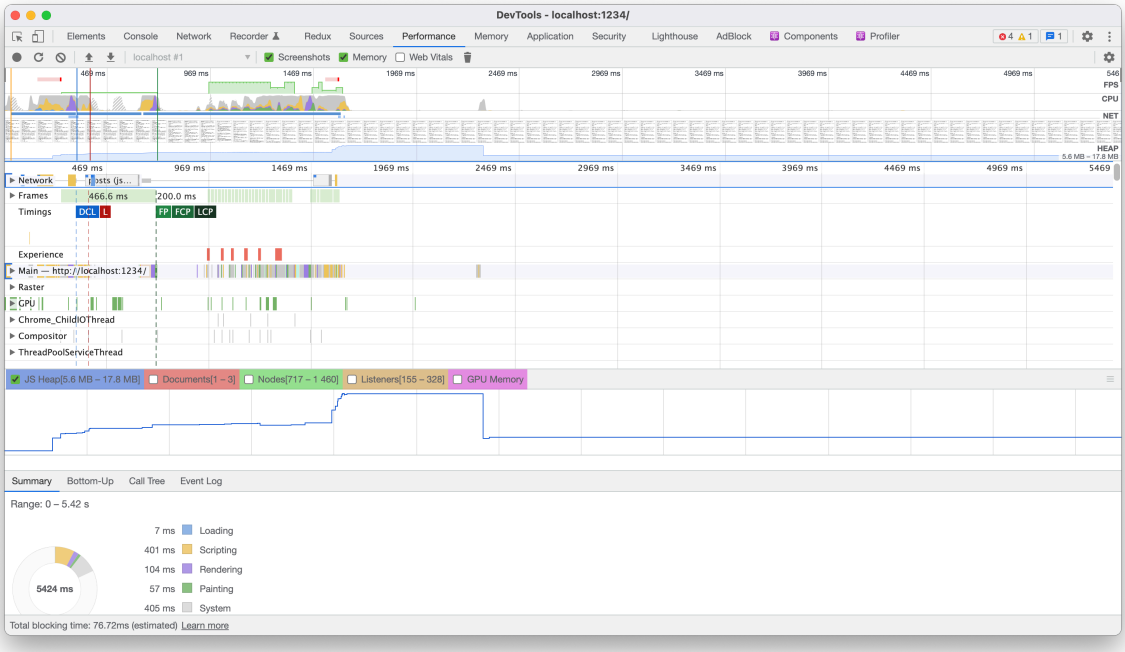

Рисунок 6 – Вкладка продуктивність

Під час зберігання даних програми за допомогою Redux або Hooks споживається пам’ять пристрою користувача. Оскільки існує багато способів

збереження оптимізації з різними структурами даних і підходами, необхідний розмір для тих самих даних у Redux і Hooks може бути нерівним.

Обсяг пам’яті вимірювався для різного розміру даних, що зберігаються в масиві 10 разів, після чого обчислювалося середнє значення. Результати вимірювань представлені в таблиці 5.

Таблиця 5 – Використання пам'яті

Розмір даних (длина масиву)	Вимірювання версії з Hooks	Вимірювання версії з Redux	Вимірювання версії з Recoil
130	8.5 MB	9.1 MB	8.1 MB
150	12 MB	17.1 MB	9.8 MB
200	16.6 MB	27.1 MB	12.8 MB

Програми створені для взаємодії з користувачами, тому швидкість реакції є ще одним важливим показником з точки зору швидкодії. Однією з можливих взаємодій у програмі, про яку йдеться, є створення нової публікації, яка викликає HTTP-запит до сервера та оновлення стану, що спричиняє повторне відображення сторінки. Google Chrome DevTools дозволяє нам виміряти точний час для цієї взаємодії. На рисунку 7 показаний приклад вимірювання для створення нової дії публікації.

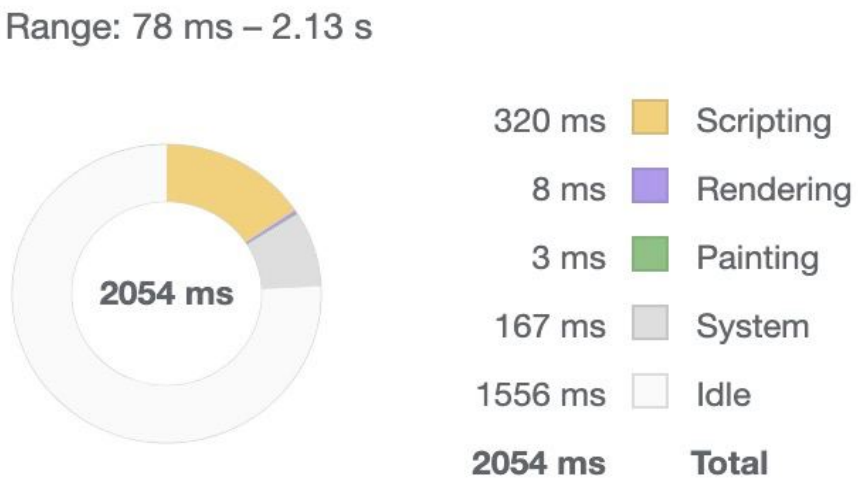

Рисунок 7 – Записана швидкодія оновлення стану

Фактичний час споживання до оновлення стану розраховується за такою формулою:

$$T_a = T_t - T_i, \quad (4)$$

де T_t– загальний час споживання,

T_i– час простою [10].

Результати вимірювань представлені в таблиці 6.

Таблиця 6 – Час операції оновлення стану

Середнє значення для версії з Hooks	Середнє значення для версії з Redux	Середнє значення для версії з Recoil
102мс	325мс	116мс

Середнє значення було отримано за допомогою проведення вимірювання взаємодії 10 разів, після чого було обчислено середнє значення.

5 АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕНЬ

Три методи керування станом для односторінкових додатків, створених за допомогою React, порівнювалися з точки зору швидкодії. Для отримання повної інформації щодо обраних методи було розглянуто та виміряно декілька показників.

В першу чергу було виміряно та проаналізовано розміри коду, необхідного для реалізації управління станом за допомогою різних методів. Отримані результати відображені на рисунку 8.

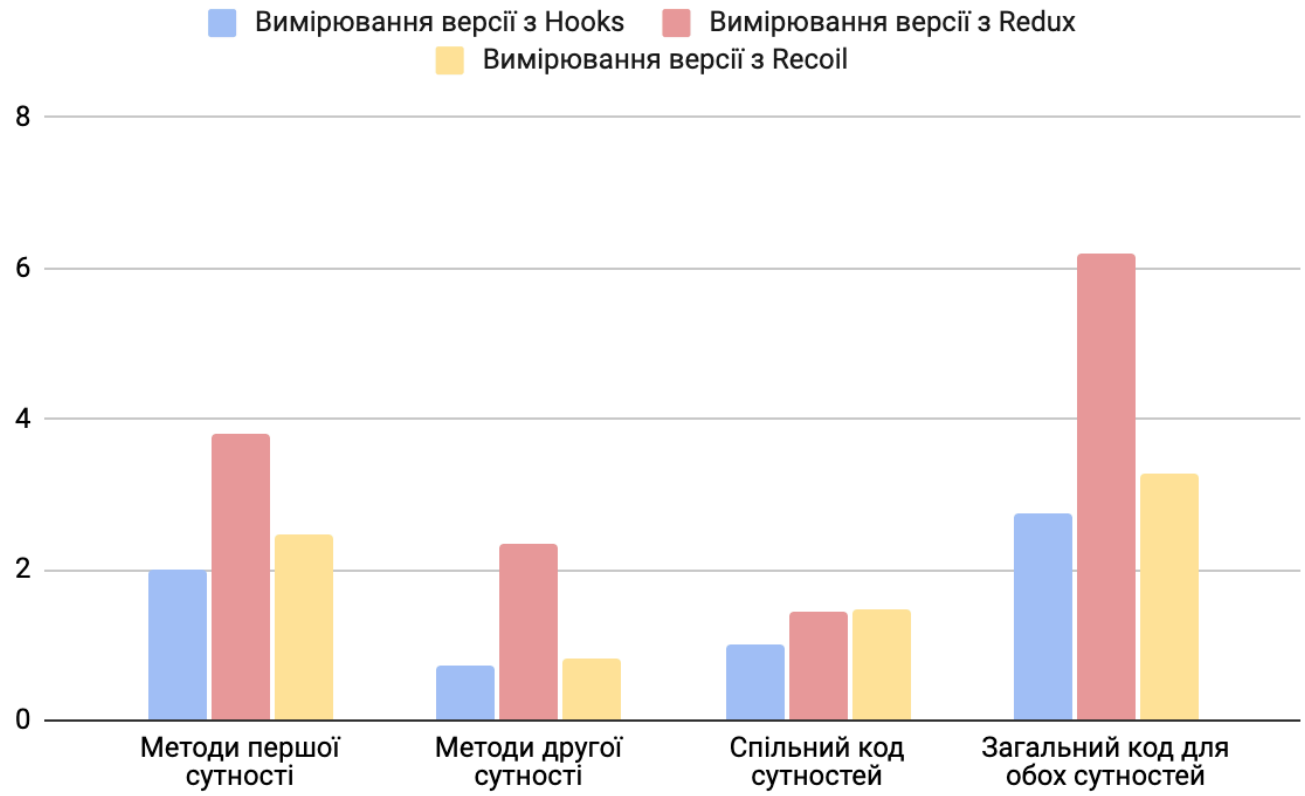

Рисунок 8 – Результати вимірювання розмірів вихідного коду

Як можна побачити на діаграмі, найбільші показники розміру коду, необхідного для реалізації керування станом, були отримані при використанні бібліотеки Redux.

Такі значення є результатом того, що використання методу Redux вимагає написання великої кількості шаблонного коду для того, щоб зберегти відповідний архітектурний підхід в управлінні станом додатків. Як методи першої сутності, так і другої, у випадку з Redux вимагали найбільшого обсягу коду для реалізації.

Беручи це до уваги, найбільший результат у обсязі загального коду для обох сутностей є закономірним.

Для реалізації методів першої та другої сутності за допомогою React Hooks та Recoil знадобилася майже однакова кількість коду. Як і Redux, метод Recoil запроваджує нові сутності, які необхідно реалізувати. Утім, для реалізації цих сутностей у випадку з Recoil знадобилося значно менше коду аніж для Redux.

Іншим важливим покажчиком є безпосередньо розмір коду у виробничій збірці веб-застосунку. На рисунку 9 зображено діаграму з результатами відповідних вимірювань.

Рисунок 9 – Результати вимірювання розмірів коду у виробничій збірці

Незважаючи на те, що в попередньому вимірюванні найгірший результат був у бібліотеки Redux, отримані результати щодо розміру коду у виробничій збірці показують, що саме метод Recoil потребує найбільшої кількості пам’яті.

Оскільки React Hooks не потребують встановлення окремих додаткових пакетів, то їхній показник є найнижчим. Тоді як для Redux та Recoil необхідно встановити безпосередньо ці бібліотеки, та ще дві додаткові у випадку з Redux:

– react-redux;
– redux-thunk.

Ці бібліотеки необхідні для забезпечення коректної роботи бібліотеки Redux з бібліотекою React і використовуються в більшості проектів, хоча бібліотека redux-thunk має інші альтернативи. Однак отриманий результат не був очікуваним.

Отримані дані також дозволили нам розрахувати інші метрики, які стосуються розмірів вихідного коду.

На рисунку 10 зображено результати обчислення дельти коду базових методів. Це відсоткове значення, яке показує вартість впровадження нових сутностей. Це важливий показник, оскільки складні веб-додатки керують великою кількістю сутностей, і його слід враховувати при розробці архітектури програми.

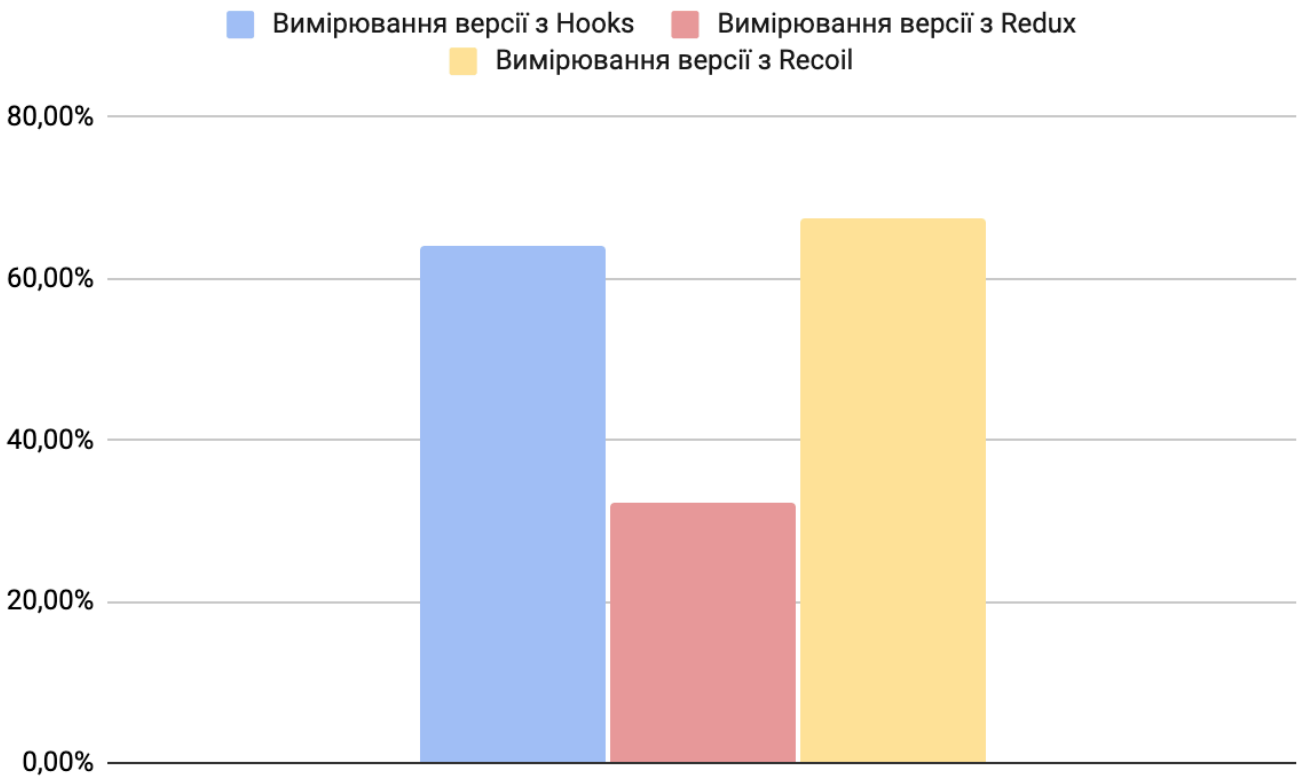

Рисунок 10 – Результати обчислення дельти коду базових методів

Як можна побачити з результатів, у версії з Redux зменшення кількості коду для реалізації базових методів подальших сутностей в проекті майже в два рази нижче ніж у версії з React Hooks чи Recoil.

Такий результат був отриманий через те, що для реалізації архітектури методу Redux необхідно впровадити усі концепції, якими цей метод оперує, тоді як, наприклад, у Recoil це не є обов'язковим.

Також було обчислено відсоток повторно використаного коду для обох сутностей для кожного з методів управління станом. Результати наведені на рисунку 11.

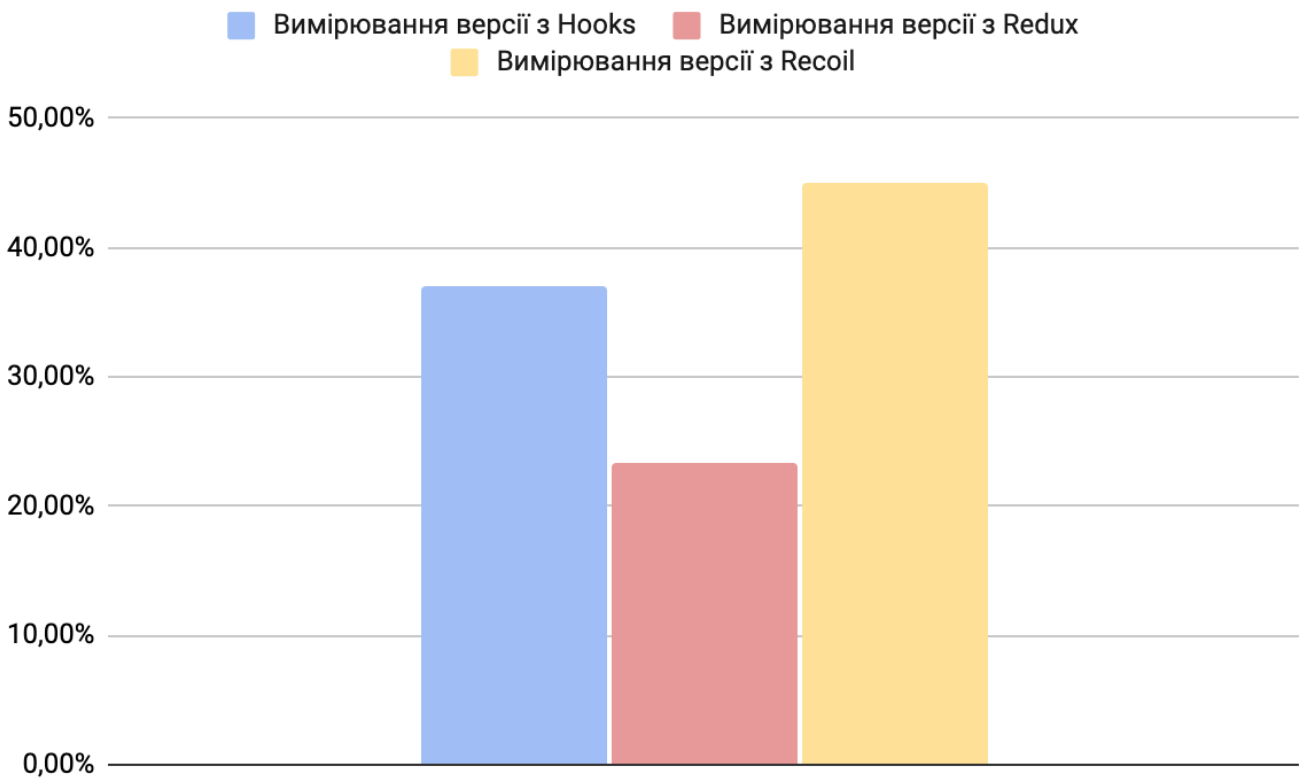

Рисунок 11 – Відсоток повторно використаного коду для обох сутностей

Результати дослідження показують, що у версіях з React Hooks та Recoil відсоток повторно використаного коду вищий. Це може свідчити про те, що в довгостроковій перспективі обслуговування цього коду буде легшим, порівняно з Redux.

На цю метрику однозначно можна легко вплинути у версії Redux за допомогою використання різних сторонніх бібліотек, але таким чином збільшиться кінцевий розмір пакета. У випадку, коли в проекті, що розробляється, існує багато сутностей, використання додаткових пакетів може виявитися більш дешевшим рішенням з точки зору розміру виробничої збірки. Однак достеменно невідомо, як

такий підхід з оптимізації вплине на швидкодію програми та на досвід розробників, що також є цінним показником у процесі розробки проекту.

З іншого боку, спроби максимізувати повторне використання коду може призвести до високого показника зв'язності коду, що не є гарним варіантом для великих проектів.

Результати обчислення метрики темпів зростання нових сутностей зображено на рисунку 12.

Рисунок 12 – Результати обчислення темпу зростання коду нових сутностей

Отримані результати цілком передбачувані після розрахунку всіх попередніх показників. Підхід Redux показав гірші результати для кожного показника, і основні причини цього:

– ведення таких понять, як дії, редуктори, проміжне програмне забезпечення;

– багато шаблонів.

У той же час метод Recoil показав найнижчий результат. Оскільки цей показник обчислюється у відсотках, то можна зробити висновок, що для реалізації

методів першої сутності у проекті необхідно значно більше коду, адже також відбувається ініціалізація необхідних об’єктів для коректного використовування цього методу.

Далі було обчислено кількість пам’яті необхідної для зберігання однакової кількості даних за допомогою різних методів. Результати представлено у вигляді діаграми, яку зображено на рисунку 13.

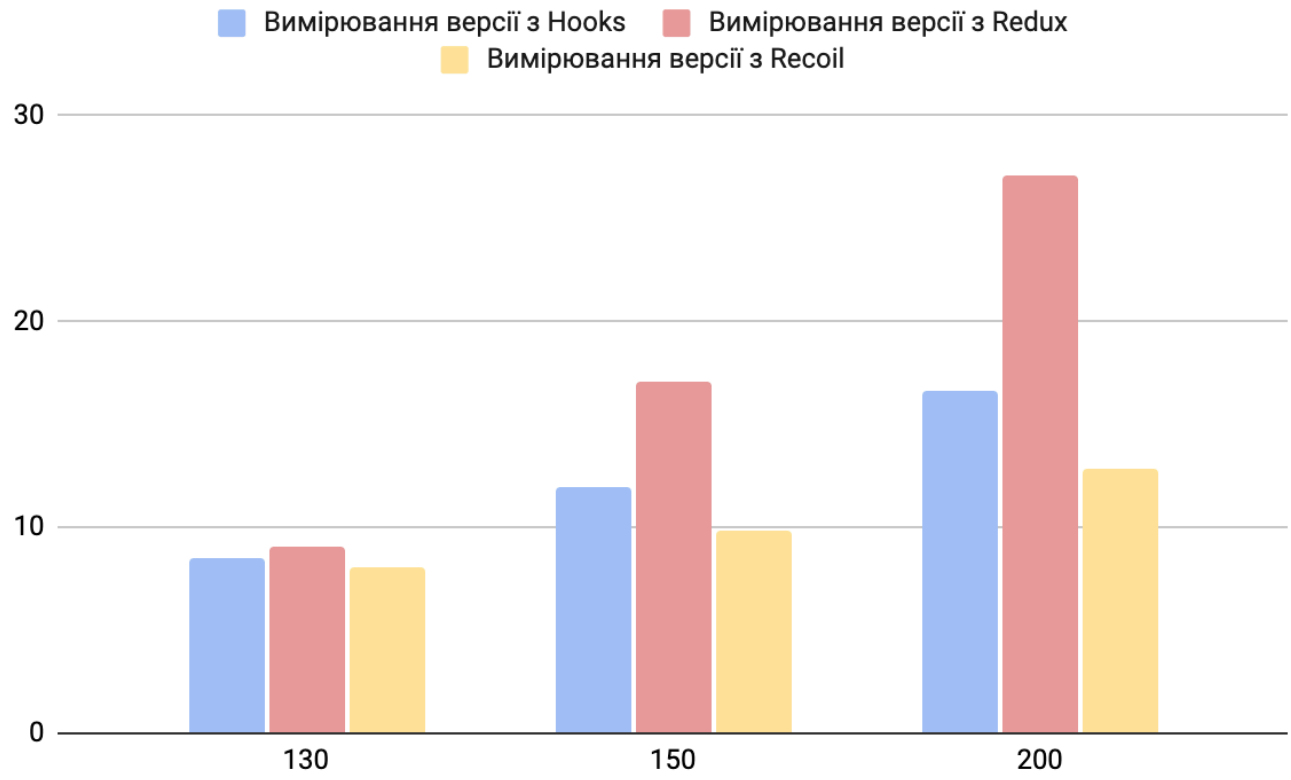

Рисунок 13 – Кількість пам’яті для зберігання однакової кількості даних

Різниця у використанні пам’яті не була помітна на невеликих обсягах даних (менше 120 елементів в масиві), тому було розглянуто та виміряно результати для більшої кількості даних.

Як можна побачити на діаграмі, кількість використаної пам’яті найшвидше зростає при використанні методу Redux, тоді як для інших методів ці показники загалом схожі. Recoil показав найкращий результат, що може бути результатом якихось додаткових оптимізацій, які відбуваються всередині.

Також було виміряно час, який потрібен для того, щоб відбулася операція оновлення стану. Результати представлені у вигляді діаграми на рисунку 14.

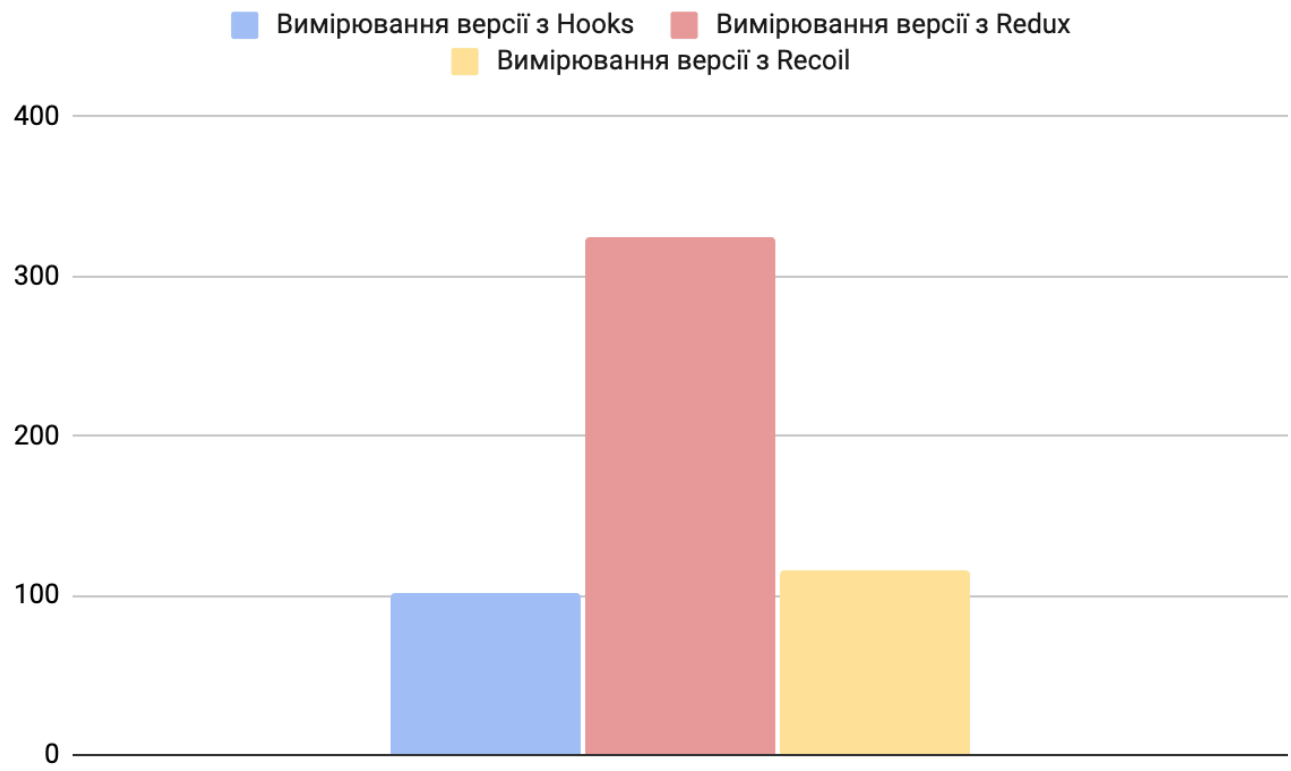

Рисунок 14 – Час оновлення стану додатку

На основі отриманих даних можна зробити висновок, що прості операції в підході Redux займають більше часу, ніж при використанні таких методів, як React Hooks чи Recoil.

У цілому, серед розглянутих методів для управління станом в односторінкових веб-додатках метод Redux показав найгірший результат майже для кожної метрики, що були розглянути в цьому дослідженні.

Завжди є місце для додаткової оптимізації, але слід враховувати обсяг додаткової роботи та досліджень, необхідних для використання того чи іншого методу.

Результати дослідження «Comparison of web application state management tools» [12] показали, що Redux є одним з найшвидших серед інших менеджерів станів, які розглядалися в цьому дослідженні. Оскільки результати поточного дослідження показують, що Redux найповільніший з точки зору швидкодії серед

усіх розглянутих методів, можна зробити висновок, що методи Ngrx і Ngxs, які підходять для фреймворку Angular, не оптимізовані з точки зору швидкодії.

Беручи до уваги той факт, що React є однією з найшвидших бібліотек для створення користувальницького інтерфейсу, що було доведено у відповідному дослідженні «DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue, and Svelte» [10], стає очевидним, що у разі потреби високої продуктивності React та React Hooks можуть стати вибором номер один для розробки односторінкових додатків.

Отримані результати дослідження є цінними у практичній діяльності, адже можуть слугувати показником степені оптимізації кожного з розглянутих методів з точки зору швидкодії.

За допомогою метрик, що були виміряні в цьому дослідженні, можна скласти картину того, як кожен з методів вплине на веб-застосунок: як буде збільшуватись кількість коду, чи швидко застосунок буде реагувати на дії користувача та інші.

Також є багато можливостей для продовження цього дослідження, як-то:

– дослідження інших методів управління станом в односторінкових додатках;

– дослідження на більш складних веб-застосунках, у яких існують функції, які використовуються у різних частинах додатку, наприклад, сповіщення чи логування.

Також можливе подальше використання отриманих значень для створення якогось уніфікованого покажчика ефективності методу управління станом з точки зору швидкодії.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи була досліджена швидкодія односторінкових застосунків в залежності від різних методів управління станом на прикладі React.

Для цього було:

– розглянуто методи управління станом в односторінкових веб-застосунках, які створені за допомогою React;

– проаналізовано та обрати методи управління станом для проведення дослідження;

– проаналізовано існуючі підходи до вимірювання швидкодії веб-застосунків;

– проаналізовано можливість впливу методів для управління станом на швидкодію веб-застосунків;

– визначено метрики, які були використані для проведення експерименту та оцінювання;

– визначено функціональні вимоги та розроблено веб-застосунок, який було використано для проведення експерименту;

– виміряно та підраховано значення обраних метрик для кожного з методів управління станом за допомогою панелі інструментів для розробників в браузері Chrome;

– порівняно та проаналізовано отримані дані з використанням таблиць та діаграм.

В рамках дослідження була проведена низка тестів і вимірювань, та були отримані результати щодо впливу кожного з методів управління станом на швидкодію.

Отже, можна зробити висновок, що підхід Redux вимагає багато шаблонів, щоб реалізувати правильний потік даних і слідувати основним рекомендаціям і концепціям. Крім того, Redux також вимагає встановлення бібліотеки react-redux, а також бібліотеки для створення асинхронних дій, щоб зробити можливими HTTP–

запити. Це призвело до гірших показників швидкодії порівняно з підходами React Hooks та Recoil.

Створення логіки потоку даних у Redux складніше і вимагає більше коду. Крім того, Redux знадобилося більше пам’яті для зберігання такої ж кількості даних у порівнянні з іншими методами.

Основна операція зміни стану забирала найбільше часу в підході Redux, що може негативно вплинути на користувацький досвід.

Методи Recoil та React Hooks показали майже однакові результати для всіх метрик. Втім, розмір виробничої збірки при використанні методу Recoil виявився занадто великим.

Перед тим як використовувати метод Recoil необхідно або впевнитися, що розмір вихідного коду у виробничій збірці є прийнятним, або стурбуватися забезпеченням оптимізації в інструменті для збірки веб-додатку.

Підводячи підсумок, можна зробити висновок, що підхід React Hooks більш оптимізований з точки зору продуктивності, ніж підхід Redux.

Беручи до уваги той факт, що дослідження було проведено з використанням веб-застосунку з обмеженим функціоналом, який був створений безпосередньо для проведення дослідження, не можна зробити однозначний висновок щодо можливих результатів у схожому дослідженні, але з використання більш складного веб-застосунку.

Однак отримані результати показали значну дельту використаних ресурсів навіть на невеликій кількості даних залежно від підходу до управління станом. Це свідчить про те, що до процесу вибору методу управління станом у веб-застосунках потрібно підходити відповідально, адже недоліки невірно обраного методу і процес заміни можуть бути дуже дорогими.

У разі високої важливості швидкодії програми не рекомендується обирати Redux як бібліотеку керування станом в односторінкових веб-додатках, що розробляються за допомогою React.

Для підтвердження отриманих результатів і зроблених висновків або для їх спростування потрібні подальші дослідження. Для подальшого дослідження слід розглянути більш складні веб-додатки з кількома сутностями та додатковими

функціями стану всього додатка, такими як повідомлення та авторизація. Слід порівняти інші методи державного управління, щоб отримати більш чітке уявлення про поточний стан менеджерів стану з точки зору швидкодії.

Кваліфікаційна робота пройшла апробацію на наступних наукових конференціях:

– на Дванадцятій міжнародній науково-технічній конференції «Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління» (тези наведено у додатку Г);

– на 6th International Conference on Computational Linguistics and Intelligent Systems (Scopus) May 12–13, 2022, Gliwice, Poland (матеріали статті наведені у додатку Д).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. React library in the npm Registry. URL: https://www.npmjs.com/package/react (дата звернення: 14.02.2022).

2. Проніна Д. М., Кириченко І. В. Порівняння впливу Redux та React Hooks підходів на швидкодію односторінкових веб-застосункі. Дванадцята міжнародна науково-технічна конференція «Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління»: тези доп. міжнар. наук-практ. конф, 27 – 28 квітня 2022 р. Баку – Харків – Жиліна, 2022. С. 146.

3. Pronina D., Kyrychenko I. Comparison of Redux and React Hooks Methods in Terms of Performance. 6th International Conference on Computational Linguistics and Intelligent Systems (Scopus), May 12–13, 2022, Gliwice, Poland.

4. Skólski P. Single-page application vs. multiple-page application [Електронний ресурс] / Paweł Skólski. – 2016. – URL: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58 (дата звернення: 13.02.2022).

5. Single Page Applications. URL: https://appcheck-ng.com/single-page-applications/ (дата звернення: 13.02.2022).

6. M. Kaluža, K. Troskot i B. Vukelić, "COMPARISON OF FRONT-END FRAMEWORKS FOR WEB APPLICATIONS DEVELOPMENT", Zbornik Veleučilišta u Rijeci, vol.6, br. 1, str. 261-282, 2018. [Online]. https://doi.org/10.31784/zvr.6.1.19 (дата звернення: 10.04.2022).

7. Angular [Електронний ресурс]. URL: https://angular.io/ (дата звернення: 14.04.2022).

8. Vue.js — The Progressive JavaScript Framework [Електронний ресурс]. URL: https://vuejs.org/ (дата звернення: 14.04.2022).

9. Mohammadi R. Performance, and Efficiency based Comparison of Angular and React in a case study of Single page application (SPA) [Електронний ресурс] / Rajab Mohammadi // Herat University, Computer Science faculty – URL: https://www.academia.edu/download/62117344/monograph20200216-17036-1haeii6.pdf (дата звернення: 10.04.2022).

10. Levlin M. DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue, and Svelte [Електронний ресурс] / Mattias Levlin // 2020 – URL: https://urn.fi/URN:NBN:fi-fe2020051838212 (дата звернення: 10.04.2022).

11. Svelte – Cybernetically enhanced web apps. URL: https://svelte.dev/ (дата звернення: 14.04.2022).

12. K. Szymanek and B. Pańczyk, “Comparison of web application state management tools”, jcsi, vol. 20, pp. 183-188, Sep. 2021.

13. Thanh Le, Comparison of State Management Solutions between Context API and Redux Hook in ReactJS, Bachelor’s Thesis, Metropolia University of Applied Sciences, Helsinki, Finland, 2021.

14. Dudar, Z., Medovoy, A., Vorochek, O. The Experience of Designing and Application of CAD Systems in Microelectronics - Proceedings of the 9th International Conference, CADSM 2007, 2007, pp. 477–478, 4297623.

15. Jeremy Wagner, Web Performance in Action: Building Fast Web Pages, Simon and Schuster, NY, 2016, C. 314-316.

16. The State of JavaScript Survey. URL: https://stateofjs.com/ (дата звернення: 16.04.2022).

17. Kirupa Chinnathambi, Learning React: A Hands-On Guide to Building Web Applications Using React and Redux, 2nd ed., Addison-Wesley Professional, Boston, MA, 2018.

18. Sophia Shoemaker, Mark Erikson, Redux: Why It's Good For You. URL: https://www.newline.co/fullstack-react/articles/redux-with-mark-erikson/ (дата звернення: 03.03.2022).

19. React v16.8: The One With Hooks. URL: https://reactjs.org/blog/2019/02/06/react-v16.8.0.html (дата звернення: 24.02.2022).

20. Мемоізація — Вікіпедія. URL: https://uk.wikipedia.org/wiki/Мемоізація (дата звернення: 20.04.2022).

21. Recoil. URL: https://recoiljs.org/ (дата звернення: 16.04.2022).

22. State management + React – Best of JS. URL: https://bestofjs.org/projects?tags=state&tags=react (дата звернення: 16.04.2022).

23. Gruzdo, I., Kyrychenko, I., Tereshchenko, G., Shanidze, N., Metrics applicable for evaluating software at the design stage, Proceedings of the 5th International Conference on Computational Linguistics and Intelligent Systems (COLINS-2021), 2021, pp. 916-936.

24. Vasilije Vasilijevic, Nenad Kojic, Natalija Vugdelija, A New Approach In Quantifying User Experience In Weboriented Applications, in: Proceedings of the 4th International Scientific Conference ITEMA, Association of Economists and Managers of the Balkans, Serbia, 2020, pp. 25-30. doi: 10.31410/ITEMA.2020.9.

25. The Principles of OOD. URL: http://www.butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod (дата звернення: 17.04.2022).

26. Karagiannis T. Structure by Type vs Feature [Електронний ресурс] / Thanos Karagiannis // maestros.io. – 2021. – URL: https://maestros.io/structure-by-type-vs-feature (дата звернення: 19.04.2022).

27. Moqups. URL: https://moqups.com/ (дата звернення: 19.04.2022).

28. Flanagan D. JavaScript: The Definitive Guide / David Flanagan — O'Reilly and Associates, 2011. — 1093 с.

29. Parcel. URL: https://parceljs.org/ (дата звернення: 19.04.2022).

30. Bytesizematters. URL: http://bytesizematters.com/ (дата звернення: 16.04.2022).

31. Gruzdo, I., Kyrychenko, I., Tereshchenko, G., Shanidze, N., Metrics applicable for evaluating software at the design stage, Proceedings of the 5th International Conference on Computational Linguistics and Intelligent Systems (COLINS-2021), 2021, pp. 916-936.

32. Національний стандарт України. Інформація та документація. Бібліографічне посилання. / Київ ДП «УкрНДНЦ» URL: http://lib.pu.if.ua/files/dstu-02-2015.pdf (дата звернення: 08.03.2022).

33. ДСТУ 7152:2010. Видання. Оформлення публікацій у журналах і збірниках. Київ, 2010. 16 с. (Інформація та документація).