Міністерство освіти і науки України

Український державний університет науки і технологій

Факультет Комп'ютерні технології та системи
Кафедра Комп'ютерні інформаційні технології

Пояснювальна записка

до кваліфікаційної роботи
магістра

на тему: «Аналіз ресурсоємності веб-додатків розроблених за допомогою React та React-native»

за освітньою програмою Інженерія програмного забезпечення
зі спеціальності: 121 Інженерія програмного забезпечення

Виконав: студент групи «ПЗ2321» _____ /Кирило ДОРОГОКУПЛЯ/

Керівник: _____ /доц. Олександр ІВАНОВ/

Нормоконтролер: _____ /Світлана ВОЛКОВА/

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань
Студент _____

Дніпро – 2025 рік

Ministry of Education and Science of Ukraine

Ukrainian State University of Science and Technologies

Faculty Computer technologies and systems
Department Computer information technology

Explanatory Note

to Master's Thesis

on the topic: «Analyzing the resource intensity of web applications developed using React and React-native»

according to educational curriculum **12 software engineering**
in the Speciality: **121 software engineering**

Done by the student of the group PZ2322: / Kyrylo DOROHOKUPLIA/

Scientific Supervisor: /Oleksandr IVANOV/

Normative controller: /Svitlana VOLKOVA/

Dnipro – 2025

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: Комп'ютерних технологій і систем
Кафедра: Комп’ютерні інформаційні технології
Рівень вищої освіти: магістр
Освітня програма: Інженерія програмного забезпечення
Спеціальність: Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ____КІТ
_____ Вадим ГОРЯЧКІН
_____ грудня 2023 р.

ЗАВДАННЯ

На кваліфікаційну роботу ____Магістр____
студенту Дорогокуплі Кирилу Олександровичу____.
1. Тема дипломної роботи: «Аналіз ресурсоємності веб-додатків розроблених за допомогою React та React-native».
Керівник роботи: Іванов Олександр Петрович
затверджені наказом 1186 ст від 29.12.2024 року
2. Строк подання студентом роботи ___.01.2025 року
3. Вихідні дані до дипломної роботи:
_____.
4. Зміст пояснювальної записки (перелік питань до розробки):
4.1. Аналітична частина: Аналіз сучасного стану дослідження за науковими літературними джерелами;
4.2. Основна частина: Часове дослідження роботи веб-додатків, розроблених за допомогою **React** та **React Native**.;
5. Перелік демонстраційного матеріалу:
5.1. презентація;
5.2. демонстраційне відео.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз сучасного стану дослідження проблеми за науковими літературними джерелами	01.09.23 – 01.10.23	10%
2	Аналіз сучасного стану програмно-апаратного забезпечення, яке потребує вдосконалення для вирішення проблем дослідження	01.10.22 – 10.10.22	
3	Постановка задачі, технічне завдання	01.09.23 – 15.09.23	30%
4	Розробка інструментальних засобів дослідження	15.09.23 – 01.11.23	
5	Виконання досліджень	01.10.23 – 01.11.23	60%
6	Оформлення пояснювальної записки	01.11.23 – 01.01.24	
7	Розробка демонстраційних матеріалів	10.01.24 – 15.01.24	100%
8	Подання кваліфікаційної роботи до кафедри	18.01.24	
9	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	25.01.24	

Студент: _____ Кирило ДОРОГОКУПЛЯ

Керівник роботи: _____ доц. Олександр ІВАНОВ

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи магістра складається з 73 стр., 56 рис. , 37 табл., 3 додатків, 11 джерел.

Об'єктом дослідження ресурсоємність веб-додатків, розроблених за допомогою React та React Native..

Мета роботи: дослідити ресурсоємність веб-додатків, створених за допомогою фреймворків React і React Native. Провести заміри часу виконання основних операцій (додавання, видалення, пошук, сортування) та оцінити обсяг використаної пам’яті. Надати рекомендації щодо вибору інструментів розробки залежно від потреб проєкту, а також провести аналіз переваг і недоліків кожного з підходів.

Методика дослідження: теоретичний аналіз наукових і літературних джерел, тестування, вимір часу виконання операцій, оцінка використання пам’яті в різних сценаріях.

Перелік ключових слів: ресурсоємність, React, React Native, веб-додатки, фреймворки, продуктивність, оптимізація.

ЗМІСТ

ВСТУП

В сучасному світі інформаційних технологій все більшої актуальності набуває питання оптимізації ресурсоємності веб-додатків. Веб-додатки, розроблені з використанням сучасних JavaScript-бібліотек, таких як React та React Native, стають основними інструментами для створення інтерактивних інтерфейсів та кросплатформних рішень.

React та React Native забезпечують високий рівень продуктивності та зручності розробки завдяки використанню компонентного підходу та віртуального DOM. Проте їх використання породжує питання ефективності використання апаратних ресурсів, особливо в умовах роботи з великими обсягами даних та виконанням складних операцій.

Мета цього дослідження — аналіз ресурсоємності веб-додатків, розроблених на основі React та React Native, а також встановлення їхніх переваг і недоліків у контексті продуктивності.

У рамках роботи проводиться:

- порівняння часу виконання ключових операцій (додавання, видалення, пошуку, сортування) з різними обсягами даних (500, 1000, 10000 елементів);
- аналіз використання пам’яті;
- виявлення залежності продуктивності від архітектури додатків.

Основна увага приділяється практичному застосуванню результатів дослідження, що дозволяє не лише отримати теоретичні знання про можливості React і React Native, але й визначити оптимальні підходи до розробки ефективних веб-додатків.

Таким чином, результати цього дослідження спрямовані на надання практичних рекомендацій щодо вибору технології залежно від вимог до продуктивності та конкретних умов використання.

РОЗДІЛ 1 АНАЛІЗ СУЧАСНОГО СТАНУ ДОСЛІДЖЕННЯ ЗА НАУКОВИМИ ЛІТЕРАТУРНИМИ ДЖЕРЕЛАМИ

1.1 Основні поняття та принципи розробки веб-додатків

Розробка веб-додатків є одним із ключових напрямків сучасних інформаційних технологій, який поєднує технології фронтенд і бекенд для створення динамічних інтерфейсів та інтерактивного функціоналу.

Основні поняття веб-додатків:

– веб-додаток — це програмне забезпечення, яке працює через веб-браузер і забезпечує інтерактивність користувацького досвіду. На відміну від традиційних веб-сайтів, веб-додатки орієнтовані на виконання завдань, таких як введення даних, обробка інформації або комунікація з сервером у реальному часі;

– клієнт-серверна архітектура — це основа веб-додатків, яка передбачає взаємодію між клієнтом (інтерфейсом користувача) та сервером (обробкою даних і бізнес-логікою);

– фронтенд — частина веб-додатку, яка відповідає за взаємодію з користувачем. Основними технологіями фронтенду є HTML (структура), CSS (візуальний стиль) і JavaScript (логіка та інтерактивність);

– бекенд — серверна частина, що відповідає за обробку даних, зберігання інформації та виконання запитів. Використовуються такі мови, як Python, Java, Node.js, і сервери баз даних, такі як MongoDB або PostgreSQL.

Основні принципи розробки веб-додатків:

– компонентний підхід – компоненти є незалежними блоками, які забезпечують повторне використання коду. У бібліотеках і фреймворках, таких як React, Vue.js, або Angular, компоненти служать основою для побудови додатків;

– модульність – додатки створюються як сукупність модулів, кожен із яких відповідає за певну функціональність. Це сприяє поліпшенню підтримки коду та його масштабованості;

– кросплатформеність – сучасні веб-додатки орієнтовані на роботу на різних пристроях та операційних системах, завдяки адаптивному дизайну та технологіям, таким як Progressive Web Apps (PWA);

– відокремлення логіки – архітектури MVC (Model-View-Controller) та MVVM (Model-View-ViewModel) дозволяють чітко розділити бізнес-логіку, інтерфейс і обробку даних;

– оптимізація продуктивності – використання кешування, оптимізація запитів до серверу та зменшення розміру ресурсів (зображень, стилів, скриптів) допомагають забезпечити швидке завантаження сторінок;

– безпека – застосування практик безпеки, таких як шифрування даних, обмеження доступу до серверу та захист від атак (наприклад, CSRF, XSS), є важливими для забезпечення конфіденційності даних користувачів;

– відповідно до викладеного, розробка веб-додатків вимагає не лише знань базових технологій, але й розуміння принципів, які допомагають створювати масштабовані, безпечні та продуктивні системи.

1.2 Особливості архітектури та функціональності React

React — це JavaScript-бібліотека, створена для розробки інтерфейсів користувача (UI). Вона є однією з найпопулярніших бібліотек для фронтенд-розробки завдяки своїй ефективності, гнучкості та широкій спільноті підтримки. React використовується для створення компонентно-орієнтованих додатків, що дозволяє розробляти масштабовані та повторно використовувані елементи інтерфейсу.

Основні особливості архітектури React:

– компонентний підхід React будує додатки з окремих компонентів. Компоненти — це незалежні модулі, які мають власну логіку та стан. Вони можуть бути простими (функціональними) або складними (класовими). Це дозволяє створювати повторно використовувані блоки інтерфейсу;

– віртуальний DOM (Virtual DOM) Замість роботи напряму з реальним DOM, React використовує віртуальний DOM для обробки змін. Це значно

підвищує продуктивність додатків, оскільки зміни в DOM відбуваються лише там, де це необхідно;

- односпрямований потік даних (One-way Data Binding) Дані передаються від батьківського компонента до дочірнього через властивості (props). Це спрощує управління станом додатка, робить його передбачуваним і зручним для тестування;
- JSX (JavaScript XML) React використовує JSX як синтаксис, що дозволяє комбінувати JavaScript та HTML у одному файлі. Це робить код більш декларативним і читабельним;
- розширюваність через сторонні бібліотеки React сам по собі зосереджений на створенні UI, але його можна інтегрувати з іншими інструментами, такими як Redux (управління станом), React Router (маршрутизація) або Styled Components (CSS-in-JS), для розширення функціональності;

Основні можливості React:

- динамічний рендеринг React дозволяє змінювати елементи інтерфейсу в реальному часі залежно від стану або дій користувача. Це досягається за допомогою методів життєвого циклу (componentDidMount, componentDidUpdate) або хуків (useEffect, useState);
- управління станом React пропонує локальне управління станом через хуки (наприклад, useState, useReducer). Для складних додатків можна використовувати зовнішні бібліотеки для глобального стану, такі як Redux або Context API;
- кросплатформеність використовуючи React Native, можна створювати мобільні додатки з використанням тієї ж архітектури та принципів, що й для веб-додатків;
- SEO-оптимізація React може використовувати серверний рендеринг (Server-Side Rendering, SSR), що підвищує швидкість завантаження сторінок і покращує SEO;

– екосистема та підтримка завдяки широкій спільноті розробників та активній підтримці з боку Facebook, React постійно вдосконалюється і має багатий набір інструментів для розробки та налагодження.

React — це потужна бібліотека для створення сучасних веб-додатків, яка пропонує просту у використанні архітектуру, гнучкість і високу продуктивність. Завдяки своїм особливостям React став ключовим інструментом у фронтенд-розробці, який забезпечує ефективне створення складних інтерфейсів користувача.

1.3 Особливості архітектури та функціональності React Native

React Native — це фреймворк для розробки мобільних додатків, який дозволяє створювати кросплатформенні додатки за допомогою JavaScript і React. Він надає можливість розробникам використовувати одну кодову базу для створення додатків під iOS та Android, що значно скорочує час та витрати на розробку.

Основні особливості архітектури React Native:

– кросплатформенність React Native забезпечує розробку додатків для декількох платформ (iOS, Android) з використанням однієї кодової бази. Це досягається через інтерфейс, що перетворює JavaScript-компоненти у нативні елементи кожної операційної системи;

– JavaScript Bridge React Native використовує міст (bridge) для зв’язку між JavaScript і нативними модулями. Код написаний на JavaScript, компілюється у нативний код платформи, що дозволяє інтегрувати функції операційних систем і використовувати їх API;

– компонентний підхід Як і React, React Native базується на компонентній архітектурі. Компоненти дозволяють створювати окремі, повторно використовувані частини інтерфейсу, що спрощує розробку та підтримку додатків;

- жестово-орієнтована розробка React Native має вбудовану підтримку для роботи з жестами та анімаціями, що дозволяє створювати інтерактивні додатки з приємним користувацьким досвідом;
- віртуальний DOM і Shadow Thread React Native не використовує DOM, але має "тіньовий потік" (Shadow Thread) для визначення макета та взаємодії з нативним UI. Це забезпечує швидке оновлення елементів на екрані.

Основні функціональні можливості React Native:

- широкий набір компонентів React Native пропонує набір стандартних UI-компонентів, які відображаються на нативні елементи платформ, такі як <View>, <Text>, <Image> тощо;
- підключення сторонніх бібліотек Завдяки екосистемі npm React Native підтримує використання численних бібліотек для розширення функціональності додатків;
- гарячий перезапуск (Hot Reloading) React Native дозволяє миттєво переглядати зміни в коді без перезапуску додатка. Це значно пришвидшує процес розробки та тестування;
- підтримка нативних модулів якщо React Native не підтримує потрібну функціональність, розробники можуть створити власні нативні модулі на Objective-C, Swift або Java та інтегрувати їх у проєкт;
- продуктивність React Native забезпечує майже нативну продуктивність завдяки використанню нативних елементів UI та оптимізації для мобільних пристроїв.

Переваги React Native:

- скорочення часу розробки завдяки повторному використанню коду.
- велика спільнота розробників та активна підтримка.
- гнучкість у налаштуванні компонентів.
- просте масштабування додатків.

Обмеження React Native:

- обмежена підтримка складних анімацій та високопродуктивних завдань;

- залежність від мосту між JavaScript і нативним кодом, що може впливати на продуктивність;
- різні платформи можуть вимагати специфічних рішень для UI.

React Native є потужним інструментом для створення мобільних додатків, який поєднує переваги кросплатформенності та нативної продуктивності. Завдяки архітектурним особливостям та багатому функціоналу цей фреймворк є популярним вибором для розробки мобільних застосунків.

1.4 Порівняльний аналіз підходів до розробки на React і React Native

React і React Native — це два популярні фреймворки, створені компанією Facebook, які використовуються для розробки інтерфейсів користувача. Попри схожість у концепціях і основах, ці інструменти мають суттєві відмінності у підходах до розробки, функціональності та застосування.

Спільні риси React і React Native:

- компонентна архітектура Обидва фреймворки базуються на концепції компонентів, які є повторно використовуваними блоками інтерфейсу;
- JSX (JavaScript XML) Для опису структури інтерфейсу використовується JSX, що дозволяє поєднувати HTML-подібний синтаксис із JavaScript.
- React-концепції Такі підходи, як стан (state), властивості (props), життєвий цикл компонентів, однаково реалізовані в обох фреймворках;
- Hot Reloading Обидва інструменти підтримують гаряче перезавантаження, що дозволяє розробникам швидко переглядати зміни в коді.

Відмінності між React і React Native

Критерій	**React**	**React Native**
Призначення	Використовується для створення веб-додатків.	Використовується для створення мобільних додатків.
Платформи	Підтримує браузери (Chrome, Firefox тощо).	Підтримує iOS та Android.

Критерій	React	React Native
Компоненти	Використовує стандартні HTML-елементи (наприклад, ,).	Використовує нативні компоненти платформ, такі як <View>, <Text>, <Image>.
Мова стилізації	CSS або інструменти CSS-in-JS.	Стилізація здійснюється за допомогою JavaScript-об’єктів, що нагадують CSS.
Результат виконання	Вихідний код перетворюється на HTML, CSS та JavaScript.	Код компілюється у нативний код платформи через JavaScript-міст.
Залежність від платформи	Універсальний для веб-платформ.	Може потребувати окремих рішень для iOS та Android через різні API та функції.
Підтримка доступу до API	Використовує браузерні API (наприклад, fetch, localStorage).	Використовує нативні API платформ через бібліотеки та модулі.

Переваги і недоліки:

React

Переваги:

- великий вибір інструментів і бібліотек для розробки;
- універсальність для створення будь-яких веб-додатків.

Недоліки:

- Вимагає додаткових налаштувань для серверного рендерингу або інтеграції з іншими інструментами.

React Native

Переваги:

- Можливість створення кросплатформених мобільних додатків;

– Нативна продуктивність і доступ до нативних функцій.

Недоліки:

– Складність у підтримці однакового вигляду та функціональності на різних платформах.

1.5 Обмеження React та React-native

Кожен із представлених фреймворків має певний ряд обмежень, які слід враховувати при виборі технології для розробки веб- чи мобільного додатку.

Обмеження React:

– складність масштабування – React є лише бібліотекою для створення користувацького інтерфейсу, тому розробникам доводиться додатково інтегрувати сторонні інструменти для управління станом (Redux, MobX) чи маршрутизацією (React Router). Це може ускладнити масштабування великих проєктів;
– проблеми з пошуковою оптимізацією (SEO) – React-додатки, що рендеряться на стороні клієнта (CSR), можуть мати обмеження у видимості для пошукових систем. Хоча серверний рендеринг (SSR) може вирішити цю проблему, його інтеграція збільшує складність розробки;
– підвищена залежність від екосистеми – через велику кількість сторонніх бібліотек і пакетів розробники можуть стикатися з проблемами сумісності, підтримки чи якості таких бібліотек;
– обмеження в продуктивності – У складних додатках, що виконують велику кількість операцій у DOM, React може показувати нижчу продуктивність у порівнянні з фреймворками, оптимізованими для вузькоспеціалізованих задач.

Обмеження React Native:

– проблеми з продуктивністю – хоча React Native пропонує майже нативну продуктивність, деякі складні обчислювальні завдання або графічні ефекти можуть працювати повільніше, ніж у чисто нативних додатках;

- залежність від JavaScript-міcту – комунікація між JavaScript і нативними компонентами через міст (Bridge) може стати вузьким місцем у продуктивності, особливо у випадках частого обміну даними;
- обмежений доступ до нативних функцій – Не всі нативні API підтримуються React Native за замовчуванням. Додатковий доступ до нативного коду (Java, Kotlin, Swift, Objective-C) потребує знань мов та роботи з платформами;
- фрагментація між платформами – хоча React Native дозволяє створювати кросплатформені додатки, часто необхідно враховувати різницю у дизайні та поведінці між iOS і Android, що додає складності в розробці;
- велика кількість сторонніх бібліотек – як і у випадку з React, наявність багатьох залежностей від бібліотек може створити проблеми з їхньою підтримкою чи сумісністю.

Висновки до розділу 1

У першому розділі було проведено аналіз сучасного стану досліджень, присвячених розробці веб- та мобільних додатків, а також особливостям застосування фреймворків React та React Native

Сучасні веб-додатки орієнтовані на багатофункціональність, інтерактивність та швидкість взаємодії з користувачем. Важливими принципами їх розробки є модульність, повторне використання компонентів та дотримання адаптивного дизайну.

React, як бібліотека для розробки інтерфейсів, забезпечує гнучкість у створенні веб-додатків завдяки компонентному підходу та застосуванню віртуального DOM. React підтримує інтеграцію сторонніх бібліотек, що розширює його функціонал, але водночас вимагає від розробників високої технічної компетентності.

React Native пропонує ефективний інструментарій для створення кросплатформених мобільних додатків із використанням єдиного коду. Основні переваги включають зниження витрат часу та ресурсів, однак обмеження

продуктивності та необхідність врахування платформних відмінностей створюють певні виклики.

React більш підходить для створення складних і динамічних веб-додатків, тоді як React Native оптимальний для мобільних рішень. Вибір фреймворку залежить від специфіки завдань проєкту, вимог до продуктивності, адаптації інтерфейсу та наявності ресурсів.

Обидва фреймворки мають свої технічні обмеження, які потребують детального врахування на етапі планування.

Загалом, проведений аналіз дозволив сформувати комплексне уявлення про можливості та обмеження React і React Native, що є основою для подальшого практичного дослідження їхньої продуктивності та ресурсоємності.

РОЗДІЛ 2 ОБГРУНТУВАННЯ МЕТОДІВ ДОСЛІДЖЕННЯ

Для проведення аналізу продуктивності та ресурсоємності додатків на основі фреймворків React і React Native буде створено спеціалізовану програму для автоматизації дослідження.

2.1 Основні функціональні вимоги

– автоматизація виконання операцій: забезпечення функціональності для додавання, видалення, пошуку, сортування масивів та списків;

– вимірювання часу виконання: визначення часу, необхідного для виконання кожної операції на різних об’ємах даних;

– обчислення використаної пам’яті: отримання даних про обсяг пам’яті, використаної для виконання всього набору операцій протягом одного експерименту;

– побудова звітів та графіків: формування звітів, які демонструють результати експериментів у вигляді таблиць та графічних діаграм;

– підтримка двох платформ: забезпечення однакового функціоналу для додатків, розроблених на основі React та React Native.

2.2 Вхідні данні

Вхідні данні до віконного додатку:

- набори даних: масиви та списки з розмірами 500, 1000 та 10 000 елементів, які представляють структуру даних типу {id, name, address}.
- розроблені сценарії: операції виконуватимуться окремо для кожного набору даних на кожній платформі;
- технічні умови: інформація про середовище виконання (операційна система, обсяг пам'яті, параметри процесора).

Критерії виконання операцій:

- для видалення — видалення кожного другого елемента;
- для пошуку — пошук першого, середнього та останнього елементів;

– для сортування — використання методів Bubble Sort та Merge Sort.

2.3 Вихідні дані

Вихідні данні до віконного додатку:

– час виконання: вимірювання тривалості кожної операції на різних обсягах даних;
– витрати пам’яті: обчислення використаного обсягу пам’яті під час виконання кожної операції;
– аналітичний звіт: рекомендації щодо вибору фреймворку для розробки залежно від вимог до продуктивності та ресурсоємності.

Графіки та порівняння:

– залежність часу виконання від розміру даних для кожної операції;
– порівняння продуктивності React і React Native для кожної операції.

Висновки до розділу 2

У даному розділі було обґрунтовано методи дослідження ефективності веб-додатків, розроблених за допомогою React та React Native. Розглянуто основні функціональні вимоги до дослідження, які включають обчислення продуктивності та використання ресурсів.

Описані вхідні дані забезпечують точність і коректність експериментів, визначаючи структуру даних, обсяг і методи доступу до них, а також сценарії використання. Це дозволяє створити експерименти, які максимально відповідають реальним умовам роботи веб-додатків.

Вихідні дані, отримані в результаті досліджень, включатимуть звіти з аналізом функціональних можливостей React і React Native, порівняльний аналіз їх продуктивності та рекомендації щодо вибору технології для конкретних умов. Також планується отримання інформації про споживання пам'яті для виконання операцій із заданими обсягами даних.

Таким чином, розроблений підхід забезпечує комплексний аналіз ефективності та продуктивності веб-додатків, дозволяючи визначити оптимальні технології для різних проектів.

РОЗДІЛ 3 РОЗРОБКА ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ДЛЯ ДОСЛІДЖЕННЯ РЕСУРСОЄМНОСТІ ВЕБ-ДОДАТКІВ

Метою даного дослідження є аналіз ресурсоємності веб-додатків, створених за допомогою фреймворків React і React Native. Для цього необхідно розробити інструментальні засоби, які дозволять виконувати операції над різними структурами даних, вимірювати час виконання операцій та обсяг використаної пам’яті.

Процес проектування інструментальних засобів передбачає використання сучасних методів візуального моделювання, таких як UML-діаграми. UML (Unified Modeling Language) — це уніфікована мова моделювання, що широко використовується для проєктування і документування програмних систем. Застосування UML сприяє структуризації задач, візуалізації компонентів системи та полегшує їх інтеграцію у фінальний проєкт.

3.1 Вибір мови програмування та середовища розробки

Для розробки інструментальних засобів, які дозволяють дослідити ресурсоємність веб-додатків, було обрано мову програмування JavaScript. Це сучасна високорівнева мова, яка широко використовується для створення веб-додатків та має потужну екосистему бібліотек і фреймворків.

Основними перевагами вибору JavaScript є:

- широке використання у веб-розробці: JavaScript є стандартом для розробки інтерактивних веб-додатків, завдяки чому надає доступ до безлічі готових рішень;
- гнучкість і динамічність: мова підтримує асинхронні операції, що є критично важливим для вимірювання продуктивності та ефективності обробки даних;
- сумісність з React і React Native: ці фреймворки створені на базі JavaScript, що забезпечує їх безшовну інтеграцію з інструментами розробки та тестування.

Для розробки інтерфейсу додатків було обрано React та React Native. Обидва фреймворки використовують JavaScript і забезпечують такі можливості:

- компонентний підхід, який сприяє повторному використанню коду та спрощує тестування;
- інструменти для вимірювання часу виконання операцій і споживання пам’яті.

Таким чином, використання JavaScript у поєднанні з Node.js, React і React Native забезпечує ефективність розробки, а також гнучкість у створенні та тестуванні додатків для дослідження їх ресурсоємності.

3.2 Формалізація задачі

Формалізація задачі була представлена у вигляді діаграми діяльності на рисунку 3.1 та діаграми прецедентів на рисунку 3.2:

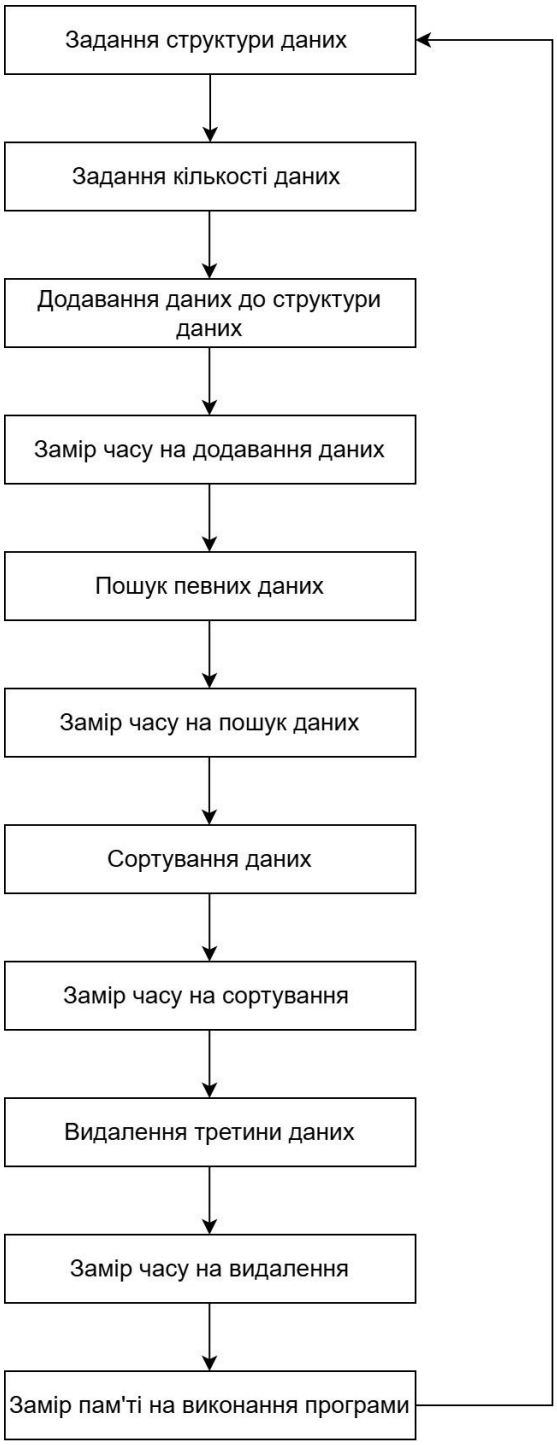

Рисунок 3.1 — діаграма діяльності аналізу ресурсоємності веб-додатків

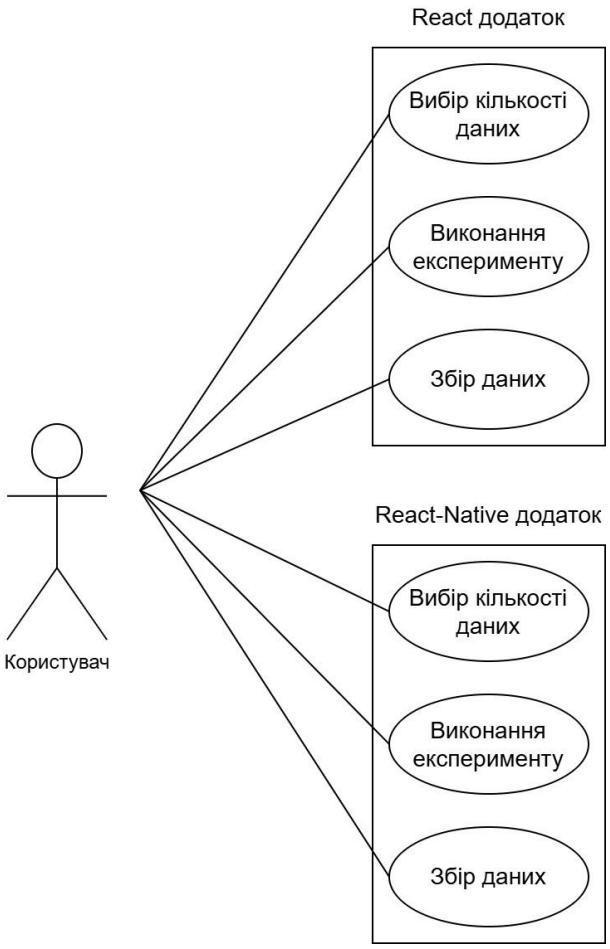

Рисунок 3.2 – діаграма прецедентів аналізу ресурсоємності веб-додатків

3.3 Особливості мови JavaScript для використання у розробці

JavaScript є однією з найпопулярніших мов програмування для створення веб-додатків завдяки своїй універсальності, широкій підтримці в браузерах та активній екосистемі. Для реалізації завдань вашого дослідження обрана ця мова через її здатність ефективно працювати як у середовищі React, так і React Native.

Основні переваги JavaScript:

- універсальність: JavaScript використовується як на стороні клієнта, так і на сервері (з Node.js). Це дозволяє створювати повноцінні застосунки, які працюють у браузері та на сервері;
- об'єктно-орієнтоване програмування: мова підтримує класи, наслідування, а також прототипне програмування, що забезпечує гнучкість у розробці великих проектів;

- асинхронність: JavaScript підтримує асинхронні операції через Promises та async/await, що особливо важливо для взаємодії з API та обробки даних у режимі реального часу;
- масштабована екосистема: існують тисячі бібліотек та фреймворків (React, Vue, Angular), які значно полегшують створення складних веб-додатків;
- кросплатформеність: використовуючи React Native, JavaScript дозволяє створювати мобільні додатки для iOS та Android із високим рівнем повторного використання коду;
- динамічна типізація: забезпечує гнучкість у роботі з різними типами даних, спрощуючи розробку, хоча й потребує обережності в складних проектах.

Особливості використання у розробці

- React надає можливість швидкої розробки та компонування інтерфейсу користувача завдяки концепції компонентів;
- React Native дозволяє зберігати більшу частину логіки спільною, розробляючи одночасно мобільні застосунки для кількох платформ;
- JavaScript забезпечує всі необхідні інструменти для тестування та автоматизації процесів вимірювання продуктивності, таких як написання скриптів для створення, пошуку, сортування та видалення елементів у структурах даних (масиви, списки, мапи).

JavaScript обрано для реалізації завдань, оскільки він оптимально підходить для створення як веб-, так і мобільних додатків. Його переваги, включаючи універсальність, асинхронність та розвинуту екосистему, роблять його незамінним інструментом у вирішенні задач дослідження.

3.4 Особливості роботи з React

Для початку роботи з React необхідно встановити середовище розробки Node.js та менеджер пакетів npm, які забезпечують підтримку бібліотек і фреймворків для розробки додатків. Наступним кроком є створення нового проекту за допомогою команди npx create-react-app, що автоматично генерує базову структуру програми.

Архітектура та компоненти

React використовує компонентний підхід до розробки, що дозволяє створювати модульний та повторно використовуваний код. Компоненти можуть бути функціональними або класовими, залежно від завдань, що виконуються. Для управління станом в компоненті використовується хук useState, а для виконання побічних дій — useEffect.

Конфігураційні налаштування проекту містять:

- папка src — основне місце для зберігання компонентів, стилів і логіки програми;
- файл App.js — головний компонент, який визначає структуру додатка.

Основні операції з даними

- додавання елементів: у React для додавання нових елементів до масивів або списків використовується функція setState, яка оновлює стан компоненту. У випадку роботи з API для надсилання даних можна використовувати fetch або бібліотеку axios;
- отримання даних: для запиту даних з API застосовується метод GET. Дані можна зберігати у стані компоненту, використовуючи хук useState;
- видалення елементів: реалізується шляхом фільтрації масиву даних за умовами, після чого стан оновлюється.

React надає інструменти для тестування компонентів і функцій, такі як бібліотека Jest для модульного тестування та React Testing Library для перевірки рендерингу компонентів. Це забезпечує надійність додатка та відповідність функціоналу початковим вимогам.

React є потужним інструментом для розробки веб-додатків завдяки компонентній архітектурі, підтримці сучасних функцій JavaScript і великій екосистемі бібліотек. Використання React у вашому проекті дозволить реалізувати ефективну роботу з даними, забезпечити гнучкість і масштабованість додатка.

3.5 Особливості роботи з React-native

Для початку роботи з React Native потрібно встановити Node.js, npm (або yarn), а також середовище розробки, наприклад, Visual Studio Code. Для створення проекту можна скористатися Expo CLI або React Native CLI, залежно від складності проекту та необхідних функцій. Команда для створення базового проекту:

npx react-native init MyProject

Або, якщо використовується Expo:

npx create-expo-app MyProject

Архітектура та компоненти

React Native використовує компонентний підхід до розробки, аналогічний React, що дозволяє створювати гнучкі та повторно використовувані компоненти. Основними особливостями є:

- використання компонентів на основі JavaScript для побудови UI;
- підтримка як функціональних, так і класових компонентів;
- для управління станом застосовуються хуки, наприклад, useState та useEffect.

Додатково, React Native надає доступ до нативних компонентів (наприклад, View, Text, Button), які рендеряться у відповідні елементи на мобільних платформах.

Основні налаштування

- файл App.js — головний компонент, у якому визначається структура програми;
- папка src — зазвичай містить компоненти, стилі, та бізнес-логіку;
- стилізація здійснюється через StyleSheet або бібліотеки (наприклад, styled-components).

Робота з даними

- додавання елементів: для додавання нових елементів використовується функція, яка оновлює стан компоненту через useState. Надсилання даних до API здійснюється за допомогою fetch або бібліотеки Axios;
- отримання даних: для отримання даних з API зазвичай використовується метод GET. Дані зберігаються у стані компоненту;
- видалення елементів: Видалення реалізується шляхом фільтрації масиву, а оновлений масив передається у стан компоненту;
- оновлення даних: оновлення виконується через зміну конкретного елемента в масиві, після чого стан оновлюється.

Тестування

Тестування React Native-додатків здійснюється з використанням інструментів:

- Jest — для модульного тестування;
- React Native Testing Library — для перевірки взаємодії компонентів і рендерингу.

Особливість запуску у веб-браузері

React Native підтримує запуск додатків у браузері за допомогою бібліотеки React Native Web. Це дозволяє розробляти і тестувати функціонал без необхідності емуляторів або фізичних пристроїв.

npm install react-native-web react-dom

Після налаштування, додаток може бути запущено у браузері, що спрощує процес тестування.

React Native дозволяє створювати кросплатформені мобільні додатки з використанням знайомого підходу React. Його можливості включають нативний доступ до функцій пристроїв, а також інтеграцію з сучасними бібліотеками для управління станом і обробки даних.

3.6 Тестування застосунків та методика вимірювання ресурсоємності

Процес тестування додатків, створених на React та React Native, мав на меті дослідити ресурсоємність під час виконання операцій із різними структурами

даних. Усі тести React Native-додатка проводилися у веб-браузері, що дозволило забезпечити однакові умови виконання для обох платформ.

Етапи тестування

Проведення тестів:

- тестувалися базові операції (додавання, видалення, пошук, сортування) для масивів, списків і словників із розмірністю 500, 1000 і 10 000 елементів;
- додатки запускалися в середовищі веб-браузера (для React Native використовувався відповідний пакет для адаптації).

Реєстрація даних:

- заміряли час виконання кожної операції та обсяг пам’яті, що споживався під час їх виконання;
- усі дані заносилися в таблиці Excel для подальшого аналізу.

Обробка результатів:

- зібрані дані було впорядковано для кожної операції;
- обчислено середні значення результатів для кожного типу операції.

Візуалізація:

- у Excel побудовано порівняльні графіки, що демонстрували залежність продуктивності операцій від розміру даних;
- додатково створено діаграми, які показували різницю у використанні пам’яті між React і React Native.

Особливості тестування у браузері

Використання браузера для тестування React Native мало такі переваги:

- уніфіковане середовище для обох платформ;
- можливість отримання точних замірів через стандартні браузерні інструменти.

Тестування підтвердило, що веб-реалізація React Native має схожі показники продуктивності з React у більшості випадків, але демонструє дещо вищі затримки через адаптацію JavaScript-коду до веб-середовища. Це дозволило зробити порівняння більш об’єктивним та релевантним

Висновки до розділу 3

У третьому розділі було розглянуто процес розробки інструментальних засобів для аналізу ресурсоємності веб-додатків, створених за допомогою React та React Native. Зокрема, висвітлено:

Обґрунтування вибору інструментів та середовища розробки:

Вибір мови JavaScript та середовищ React і React Native пояснюється їхньою популярністю, високою продуктивністю та зручністю роботи з ними.

Використання єдиної мови програмування забезпечило можливість створення крос-платформних рішень, спрощення тестування та аналізу.

Особливості реалізації:

Описано методику створення додатків, яка включала інтеграцію операцій із різними структурами даних (масиви, списки, мапи) та автоматизацію вимірювання їхньої продуктивності.

Реалізація додатків була виконана з урахуванням відмінностей у роботі React та React Native, зокрема адаптації останнього для виконання у браузері.

Тестування та аналіз:

Проведено тестування додатків, де було зміряно час виконання операцій та використання пам’яті для різних розмірів структур даних.

Зібрані результати опрацьовано у Excel, а ключові висновки візуалізовано за допомогою графіків.

Основні результати:

Додатки успішно виконували операції над структурами даних, що дозволило отримати валідні результати для аналізу.

React виявився більш продуктивним у веб-середовищі, тоді як React Native мав додаткові затримки через адаптацію коду.

Загальний висновок

Розроблені інструментальні засоби продемонстрували ефективність використання обраних технологій для виконання завдань дослідження. Процес реалізації та тестування підтвердив придатність React і React Native для роботи з великими обсягами даних, забезпечуючи відповідний рівень продуктивності. Отримані результати слугуватимуть основою для проведення порівняльного аналізу ресурсоємності цих технологій у наступних розділах.

РОЗДІЛ 4 АНАЛІЗ ЕФЕКТИВНОСТІ ТА ПРОДУКТИВНОСТІ СИСТЕМ ДОБД

У цьому розділі представлено результати дослідження ресурсоємності додатків, розроблених за допомогою React і React Native. Експерименти проводилися з різними обсягами даних та розмірами елементів для оцінки ключових показників продуктивності.

Для оцінки продуктивності додатків було створено масиви даних з такими характеристиками:

Кількість елементів:

- 500 елементів;
- 1 000 елементів;
- 10 000 елементів.

Ефективність роботи додатків оцінювалася за такими показниками:

- час додавання всіх елементів у структури даних;
- час видалення кожного другого елемента зі структур даних;
- час пошуку першого, середнього та останнього третіх елементів;
- час сортування масивів із застосуванням алгоритмів сортування:
  - бульбашкового сортування;
  - сортування злиттям;
- кількість використаної пам’яті для виконання усіх операцій.

Для дослідження використовувались масиви, списки та карти як базові структури даних. Усі результати вимірювань зберігалися в Excel, а для кожного показника продуктивності будувалися графіки для наочного порівняння роботи додатків, створених на основі React і React Native.

Аналіз результатів дозволяє зробити висновки щодо продуктивності цих технологій залежно від розміру даних, обраної структури даних та складності операцій. Одержані дані також є основою для формулювання рекомендацій щодо вибору підходящої технології для задач, пов'язаних із обробкою великих обсягів даних.

Позначення та структура графіків у розділі:

У даному розділі для відображення результатів дослідження буде використано графіки, які представлені у такій формі:

– синій графік відображає дані для веб-додатків, створених за допомогою React;

– помаранчевий графік представляє результати для веб-додатків, розроблених за допомогою React Native;

– вісь Ох позначає кількість спроб/експериментів, які були проведені під час дослідження;

– вісь Оу відображає кількість часу, необхідного для виконання операції, у мілісекундах. У випадках аналізу пам’яті вісь Оу показує обсяг пам’яті, витраченої на виконання всіх операцій, у мегабайтах.

4.1 Дослідження роботи під час опрацювання 500 елементів

Під час дослідження часу виконання операцій з обробки даних у розроблених додатках на основі React та React Native, було отримано результати вимірювань, які наведено в таблиці 4.1, 4.2, 4.3, 4.4, 4.5, 4.6 та таблиці 4.7, 4.8, 4.9, 4.10, 4.11, 4.12 :

Таблиця 4.1 – Час виконання операцій для React на основі Array

№	Add	Find	Delete	Bubble S	Merge S
1	15	0,2	0	2,40	0,3
2	17,8	0,29	0	3,50	0,2
3	19,9	0,2	0	2,50	0,2
4	12,6	0,3	0	1,00	0
5	10,1	0,19	0	1,00	0,2
6	14,2	0,3	0	1,00	0,2
7	11,6	0,2	0	1,2	0,4
8	10,7	0,2	0	1,00	0,5
9	13,4	0,3	0	1,00	0,1
10	15,50	0,1	0,1	1,1	0,09

Таблиця 4.2 – Кількість використаної пам’яті для React на основі Array

№	Memory
1	12,258
2	11,092
3	11,69
4	11,328
5	11,02
6	11,15
7	10,83
8	10,87
9	11,41
10	11,16

Таблиця 4.3 – Час виконання операцій для React на основі List

№	Add	Find	Delete	Bubble S	Merge S
1	0,9	0,5	0,1	5,6	0,70
2	0,4	0,3	0,1	4,8	0,60
3	0,3	0,3	0	5,4	0,40
4	0,3	0,4	0	5,7	0,40
5	0,2	0,3	0	2,5	0,60
6	0,3	0,2	0	2,2	0,50
7	0,3	0,2	0	2,4	0,40
8	0,3	0,5	0	3,7	0,70
9	0,3	0,4	0	2,4	0,60
10	0,2	0,2	0	2,1	0,40

Таблиця 4.4 – Кількість використаної пам’яті для React на основі List

№	Memory
1	24,811
2	24,411
3	24,434
4	24,339
5	24,717
6	24,994
7	24,897
8	25,079
9	25,223

10	25,256

Таблиця 4.5 – Час виконання операцій для React на основі Map

№	Add	Find	Delete	Bubble S	Merge S
1	0,2	0,5	0,4	4,9	0,8
2	0,1	0,2	0,1	0,8	0,5
3	0,1	0,2	0	2,9	0,5
4	0,1	0,3	0	0,7	0,3
5	0,1	0,2	0	1,8	0,4
6	0,2	0,2	0	1,5	0,5
7	0,3	0,4	0,3	1	0,5
8	0,2	0,1	0,1	0,6	0,4
9	0,2	0,2	0,1	0,7	0,3
10	0,5	0,4	0	0,8	0,5

Таблиця 4.6 – Кількість використаної пам’яті для React на основі Map

№	Memory
1	37,09
2	37,276
3	37,418
4	37,718
5	37,467
6	37,166
7	37,294
8	37,73
9	38,055
10	38,184

Таблиця 4.7 – Час виконання операцій для React-Native на основі Array

№	Add	Find	Delete	Bubble S	Merge S
1	12,40	0,1	0	2,8	0,6
2	13,6	0,1	0	2,3	0,2
3	12,9	0,1	0	3,50	0,3
4	14,2	0	0	1,9	0,2
5	13,6	0	0	1,1	0,1
6	15,7	0	0	1,1	0,1
7	11,50	0	0	1,4	0,2
8	10,6	0	0	0,9	0,2

9	12,4	0	0	2,5	0,4
10	13,8	0,1	0	1,1	0,3

Таблиця 4.8 – Кількість використаної пам’яті для React-Native на основі Array

№	Memory
1	25,198
2	24,52
3	24,977
4	25,662
5	24,675
6	24,629
7	24,988
8	24,716
9	24,977
10	26,703

Таблиця 4.9 – Час виконання операцій для React-Native на основі List

№	Add	Find	Delete	Bubble S	Merge S
1	0,8	0,2	0,1	3,7	0,4
2	0,8	0	0,1	3	0,2
3	1,1	0	0,1	7,3	0,4
4	0,9	0,1	0	6	0,3
5	0,8	0	0,1	3,2	0,2
6	0,9	0,2	0	4,6	0,3
7	0,7	0,1	0,1	5,50	0,2
8	0,6	0,1	0	3,2	0,1
9	0,8	0,1	0	4,2	0,3
10	0,7	0,1	0	4,8	0,1

Таблиця 4.10 – Кількість використаної пам’яті для React-Native на основі List

№	Memory
1	32,075
2	25,753
3	24,613
4	24,931
5	24,646
6	24,627
7	25,205
8	24,829

9	24,61
10	24,98

Таблиця 4.11 – Час виконання операцій для React-Native на основі Map

№	Add	Find	Delete	Bubble S	Merge S
1	0,2	0,2	0,1	2,90	0,5
2	0,1	0,2	0,1	0,7	0,4
3	0,2	0,4	0,1	2,90	0,8
4	0,2	0,3	0,2	5,30	0,3
5	0,1	0,2	0	3,20	0,4
6	0,1	0,3	0	1,00	0,6
7	0,1	0,2	0,1	0,60	0,3
8	0,3	0,2	0	0,60	0,3
9	0,1	0,3	0	0,60	0,3
10	0,1	0,3	0	0,60	0,4

Таблиця 4.12 – Кількість використаної пам’яті для React-Native на основі Map

№	Memory
1	25,63
2	25,63
3	24,69
4	24,98
5	25,26
6	26,1
7	25,26
8	25,24
9	25,74
10	25,96

З отриманих даних було складено порівняльні графіки.

Графік затраченого часу на додавання 500 елементів в структуру даних Array представлено на малюнку 4.1:

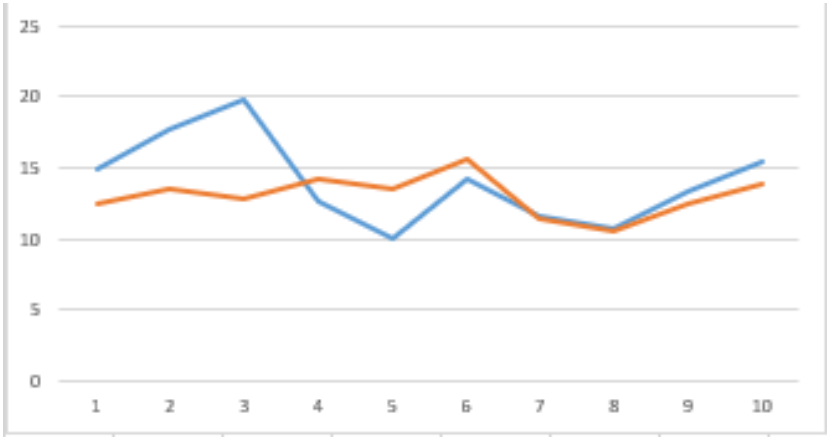

Рисунок 4.1 — Графік порівняння затраченого часу на додавання елементів

З отриманого графіку можна сказати що операції додавання елементів на React-Native займає менше часу ніж додавання на React.

Графік затраченого часу на пошук серед 500 елементів для структури даних Array представлено на малюнку 4.2:

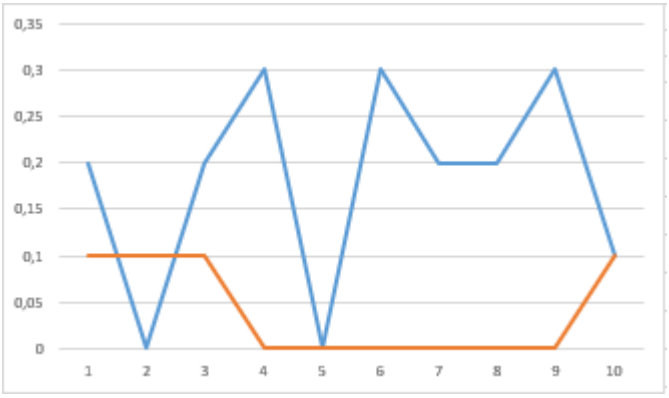

Рисунок 4.2 — Графік порівняння затраченого часу на пошук елементів

З отриманого графіку можна сказати що операції пошуку елементів на React займає більше часу ніж на React-Native.

Графік затраченого часу на видалення елементів для структури даних Array наведено на рисунку 4.3:

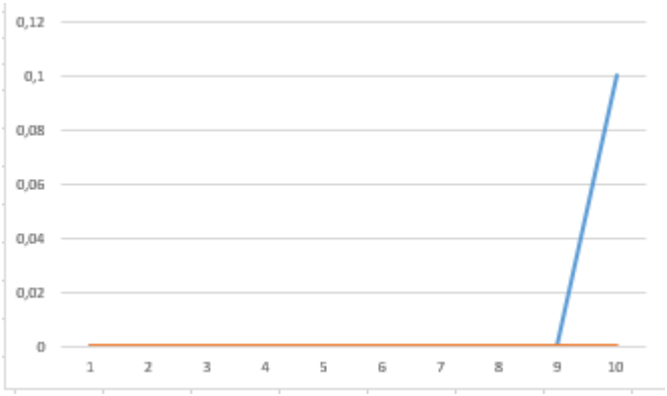

Рисунок 4.3 — Графік порівняння затраченого часу на видалення всіх елементів

З отриманого графіку можна сказати що операції видалення елементів займає приблизно однаковий час на обох фреймворках.

Графік затраченого часу на сортування бульбашковим методом в Array наведено на рисунку 4.4:

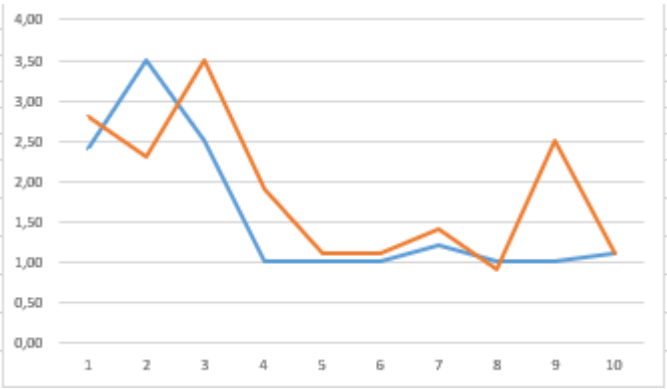

Рисунок 4.4 — Графік порівняння затраченого часу на сортування бульбашкою

З отриманого графіку можна сказати що операції сортування бульбашкою займає приблизно однаковий час, проте на React-Native – більше.

Графік затраченого часу на сортування методом злиття в Array наведено на рисунку 4.5:

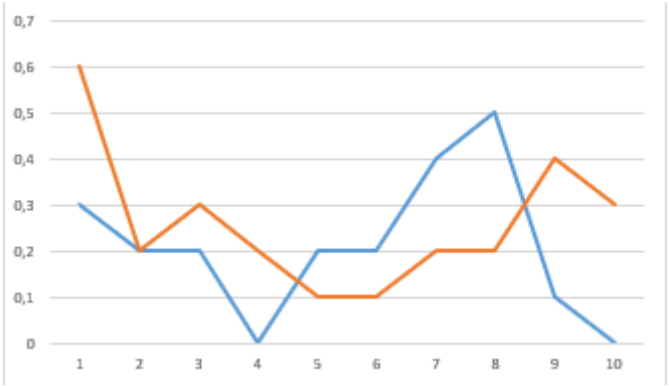

Рисунок 4.5 — Графік порівняння затраченого часу на сортуванням методом злиття

З отриманого графіку можна сказати що операції сортування елементів методом злиття займає приблизно однаковий час, графік симетричний, проте на React займає менше.

Графік порівняння витраченої пам’яті на виконання всіх операцій наведено на рисунку 4.6:

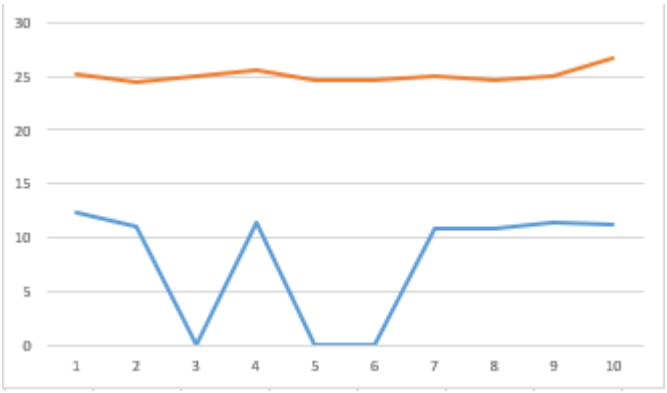

Рисунок 4.6 — Графік порівняння витраченої пам’яті

З отриманого графіку можна сказати що виконання операцій на React-Native використовує пам’яті більше ніж на React.

Графік затраченого часу на додавання 500 елементів в структуру даних List представлено на рисунку 4.7:

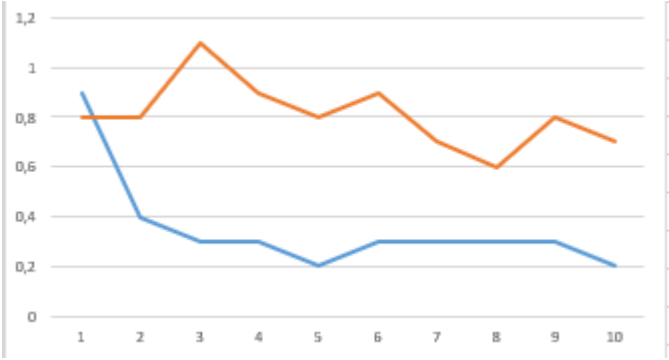

Рисунок 4.7 – Графік порівняння затраченого часу на додавання елементів

З отриманого графіку можна сказати що операції додавання елементів на React займає набагато менше часу ніж на React-Native.

Графік затраченого часу на пошук серед 500 елементів для структури даних List представлено на малюнку 4.8:

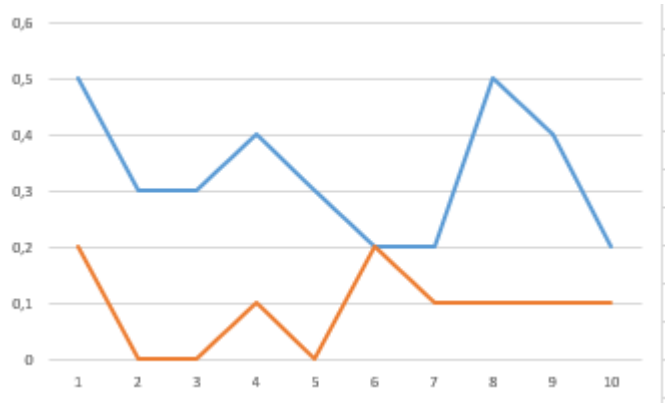

Рисунок 4.8 – Графік порівняння затраченого часу на пошук елементів

З отриманого графіку можна сказати що операції пошуку елементів на React займають більше ніж на React-Native.

Графік затраченого часу на видалення елементів для структури даних List наведено на рисунку 4.9:

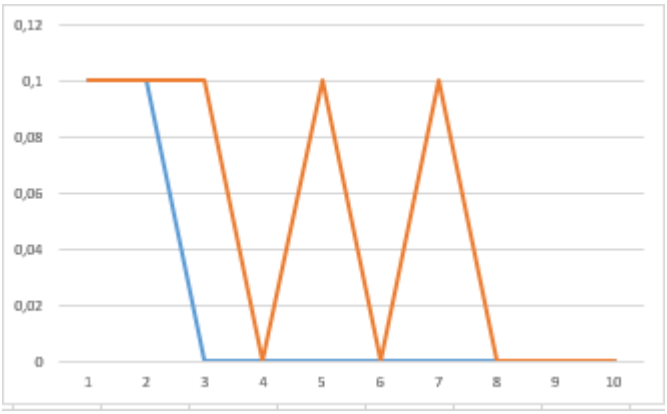

Рисунок 4.9 – Графік порівняння затраченого часу на видалення всіх елементів

З отриманого графіку можна сказати що операції видалення елементів на React-Native займає набагато більше часу ніж на React.

Графік затраченого часу на сортування бульбашковим методом в List наведено на рисунку 4.10:

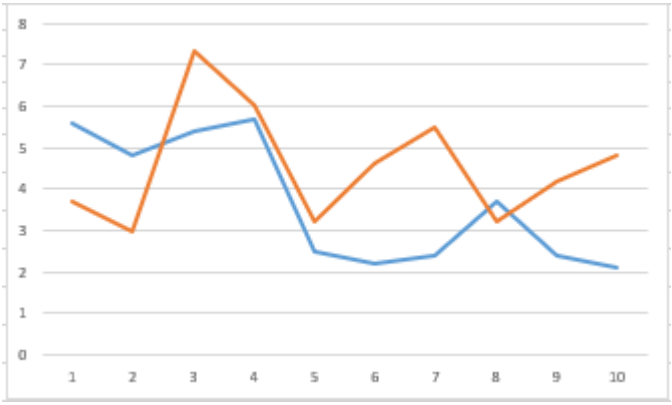

Рисунок 4.10 – Графік порівняння затраченого часу на сортування бульбашкою

З отриманого графіку можна сказати що операції сортування елементів бульбашкою займає приблизно однаковий час, проте на React – менше.

Графік затраченого часу на сортування методом злиття в Array наведено на рисунку 4.11:

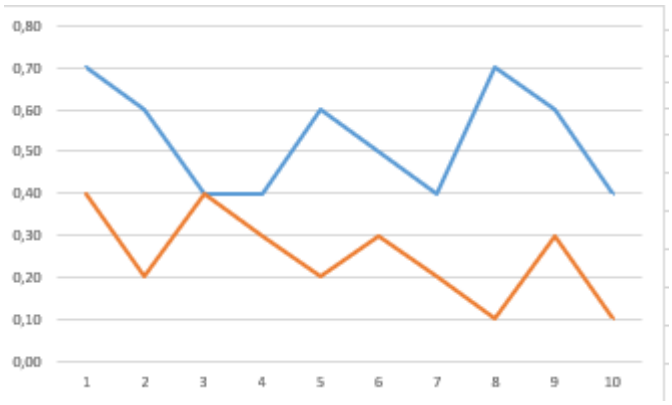

Рисунок 4.11 – Графік порівняння затраченого часу на сортуванням методом злиття

З отриманого графіку можна сказати що операції сортування елементів методом злиття на React займає більше часу в порівнянні з React-Native.

Графік порівняння витраченої пам’яті на виконання всіх операцій для List наведено на рисунку 4.12:

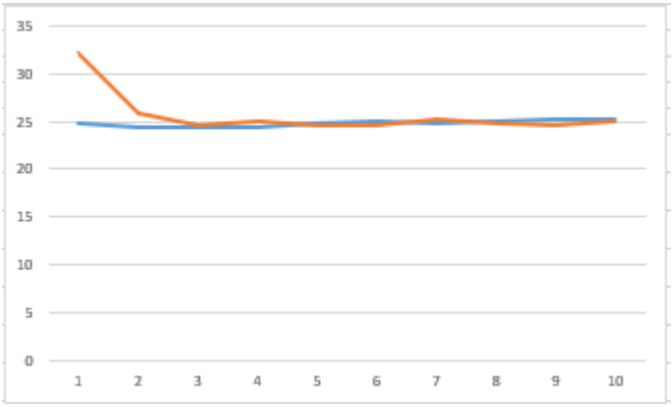

Рисунок 4.12 – Графік порівняння витраченої пам’яті

З отриманого графіку можна сказати що виконання операцій на React-Native використовує пам’яті майже стільки же як і на React.

Графік затраченого часу на додавання 500 елементів в структуру даних Map представлено на малюнку 4.13:

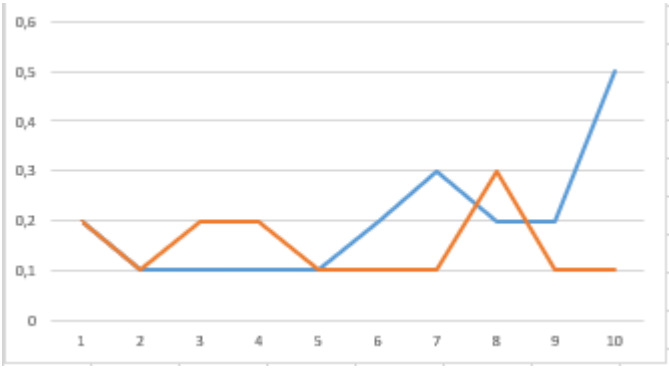

Рисунок 4.13 – Графік порівняння затраченого часу на додавання елементів

З отриманого графіку можна сказати що операції додавання елементів на React займає більше часу в порівнянні з React-Native.

Графік затраченого часу на пошук серед 500 елементів для структури даних Map представлено на малюнку 4.14:

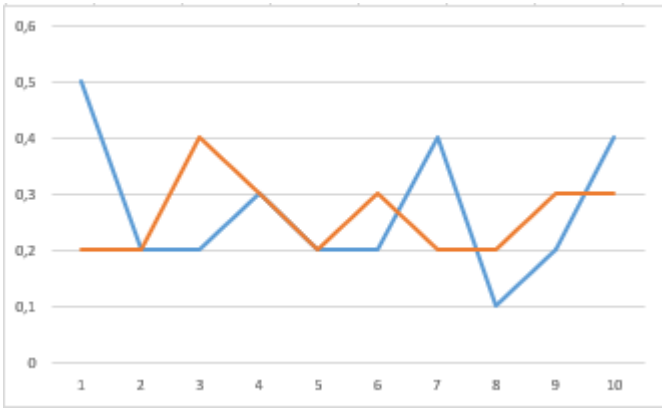

Рисунок 4.14 – Графік порівняння затраченого часу на пошук елементів

З отриманого графіку можна сказати що операції пошуку елементів на React займає менше часу в порівнянні з React-Native.

Графік затраченого часу на видалення елементів для структури даних Map наведено на рисунку 4.15:

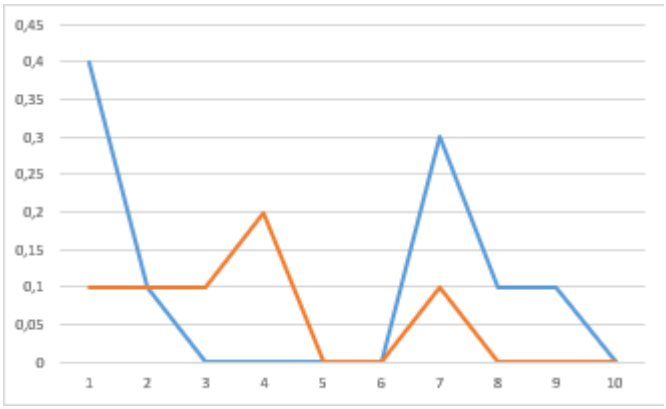

Рисунок 4.15 – Графік порівняння затраченого часу на видалення всіх елементів

З отриманого графіку можна сказати що операції видалення елементів на React займає більше часу в порівнянні з React-Native.

Графік затраченого часу на сортування бульбашковим методом в Map наведено на рисунку 4.16:

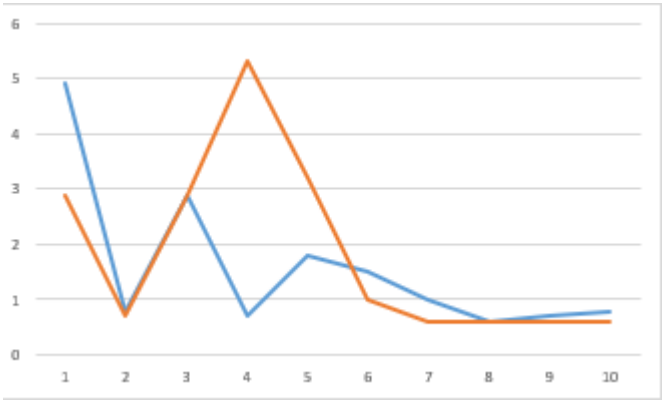

Рисунок 4.16 – Графік порівняння затраченого часу на сортування бульбашкою

З отриманого графіку можна сказати що операції сортування елементів бульбашкою на React займає менше часу в порівнянні з React-Native.

Графік затраченого часу на сортування методом злиття в Map наведено на рисунку 4.17:

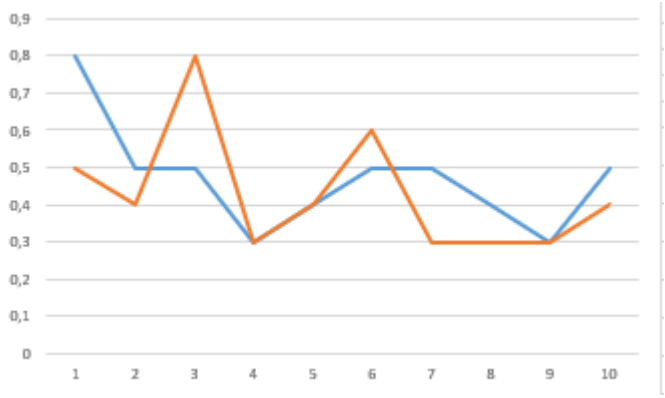

Рисунок 4.17 – Графік порівняння затраченого часу на сортуванням методом злиття

З отриманого графіку можна сказати що операції сортування елементів методом злиття на React займає більше часу в порівнянні з React-Native.

Графік порівняння витраченої пам’яті на виконання всіх операцій для Map наведено на рисунку 4.18:

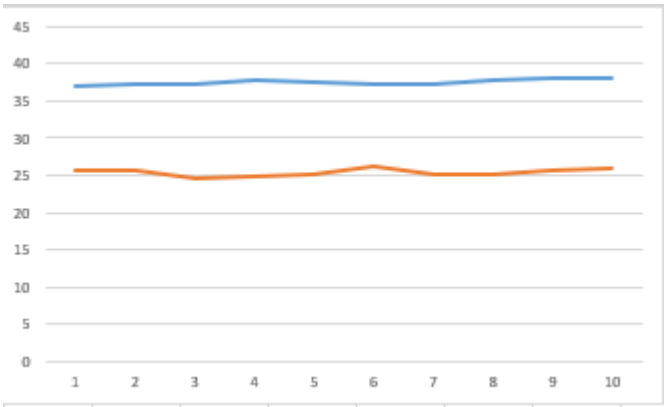

Рисунок 4.18 – Графік порівняння витраченої пам’яті

З отриманого графіку можна сказати що виконання операцій на React використовує пам’яті більше ніж на React.

4.2 Дослідження роботи під час опрацювання 1000 елементів

Під час дослідження часу виконання операцій з обробки даних у розроблених додатках на основі React та React Native, було отримано результати вимірювань, які наведено в таблиці 4.13, 4.14, 4.15, 4.16, 4.17, 4.18 та таблиці 4.19, 4.20, 4.21, 4.22, 4.23, 4.24 :

Таблиця 4.13 – Час виконання операцій для React на основі Array

№	Add	Find	Delete	Bubble S	Merge S
1	19,1	0,3	0,1	4,2	0,4
2	37,2	0,5	0	4,6	0,4
3	19,8	0,29	0	4,2	0,3
4	20,6	0,3	0	4,2	0,3
5	22,8	0,5	0	4,3	0,3
6	21,2	0,5	0	4,4	0,5
7	19,4	0,4	0,1	4,4	0,3
8	20,50	0,3	0,1	4,2	0,3
9	20,8	0,3	0,10	4,3	0,2
10	20,9	0,3	0	4,2	0,2

Таблиця 4.14 – Кількість використаної пам’яті для React на основі Array

№	Memory
1	13,054
2	12,023
3	12,906
4	12,107
5	12,552
6	11,905
7	12,786
8	12,82
9	12,814
10	12,823

Таблиця 4.15 – Час виконання операцій для React на основі List

№	Add	Find	Delete	Bubble S	Merge S
1	1,1	0,6	0	9,09	1,3
2	1,3	0,3	0	9,3	1,1
3	1,1	0,4	0	8,5	1,1
4	1,1	0,4	0	8,8	1,2
5	1	0,6	0	8,3	1,2
6	1,3	0,4	0	10,2	1,3
7	1	0,5	0	9,3	1,2
8	1,2	0,3	0	9,4	1,1
9	1	0,3	0	9,7	1,4
10	1	0,3	0	9,1	1,2

Таблиця 4.16 – Кількість використаної пам’яті для React на основі List

№	Memory
1	24,906
2	25,478
3	25,796
4	25,744
5	26,03
6	25,954
7	25,719
8	26,083
9	26,277
10	27,676

Таблиця 4.17 – Час виконання операцій для React на основі Map

№	Add	Find	Delete	Bubble S	Merge S
1	0,2	0,4	0	2,6	0,7
2	0,2	0,4	0	2,6	0,6
3	0,3	0,3	0,1	2,6	0,7
4	0,3	0,4	0	2,8	0,9
5	0,2	0,3	0,2	2,6	0,8
6	0,3	0,4	0,4	3,6	1
7	0,3	0,4	0,1	0,1	0,8
8	0,4	0,3	0	3,5	0,8
9	0,3	0,4	0,1	2,7	1
10	0,2	0,4	0,1	3	0,9

Таблиця 4.18 – Кількість використаної пам’яті для React на основі Map

№	Memory
1	38,524
2	38,537
3	38,147
4	37,887
5	38,573
6	38,306
7	38,331
8	38,777
9	38,494
10	38,66

Таблиця 4.19 – Час виконання операцій для React-Native на основі Array

№	Add	Find	Delete	Bubble S	Merge S
1	22,2	0	0	3,8	0,3
2	20,9	0	0	4,1	0,2
3	19,9	0	0,1	4,1	0,3
4	23,9	0	0,1	4,1	0,2
5	21,7	0	0,1	4,1	0,8
6	21	0	0,1	4,2	0,4
7	20,9	0	0,1	3,8	0,3
8	23	0	0,1	4	0,4
9	20,2	0	0	4,2	0,2
10	27,7	0	0,2	5,3	0,4

Таблиця 4.20 – Кількість використаної пам’яті для React-Native на основі Array

№	Memory
1	25,49
2	26,211
3	27,017
4	24,868
5	25,704
6	24,543
7	25,865
8	25,549
9	26,096
10	28,109

Таблиця 4.21 – Час виконання операцій для React-Native на основі List

№	Add	Find	Delete	Bubble S	Merge S
1	2	0,4	0	12	0,4
2	1,7	0	0,1	10,4	0,4
3	1,8	0,3	0	10,9	0,2
4	2,2	0,4	0	10,1	0,4
5	2,1	0,2	0	10,9	0,4
6	1,7	0,1	0	10,3	0,3
7	1,50	0,1	0,1	10,8	0,4
8	1,6	0,1	0,1	11,4	0,4
9	2,1	0,1	0	9,50	0,4
10	2	0,2	0,1	10,8	0,4

Таблиця 4.22 – Кількість використаної пам’яті для React-Native на основі List

№	Memory
1	24,859
2	25,456
3	24,647
4	25,397
5	25,314
6	25,856
7	24,703
8	26,296
9	25,693
10	24,765

Таблиця 4.23 – Час виконання операцій для React-Native на основі Map

№	Add	Find	Delete	Bubble S	Merge S
1	0,2	0,6	0,1	2,8	0,5
2	0,4	0,3	0	3,5	0,9
3	0,4	0,8	0,1	3	0,8
4	0,3	0,3	0	2,6	0,4
5	0,3	0,4	0,1	2,6	0,4
6	0,3	0,4	0	2,6	0,4
7	0,6	0,6	0	2,7	0,5
8	0,2	0,3	0	2,6	0,4
9	0,2	0,3	0,2	3,4	1
10	0,2	0,5	0	2,7	0,4

Таблиця 4.24 – Кількість використаної пам’яті для React-Native на основі Map

№	Memory
1	26,16
2	25,86
3	26,36
4	26,8
5	27,19
6	27,63
7	27,99
8	28,26
9	27,89
10	27,16

З отриманих даних було складено порівняльні графіки.

Графік затраченого часу на додавання в структуру даних Array представлено на малюнку 4.19:

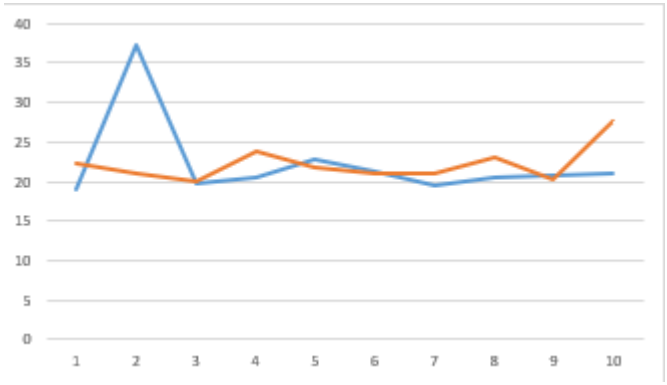

Рисунок 4.19 — Графік порівняння затраченого часу на додавання елементів

З отриманого графіку можна сказати що операції додавання елементів на React-Native займає менше часу ніж додавання на React.

Графік затраченого часу на пошук елементів для структури даних Array представлено на малюнку 4.20:

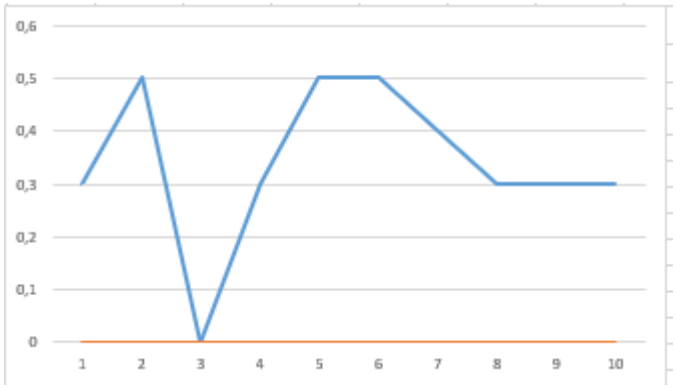

Рисунок 4.20 — Графік порівняння затраченого часу на пошук елементів

З отриманого графіку можна сказати що операції пошуку елементів на React займає більше часу ніж на React-Native.

Графік затраченого часу на видалення елементів для структури даних Array наведено на рисунку 4.21:

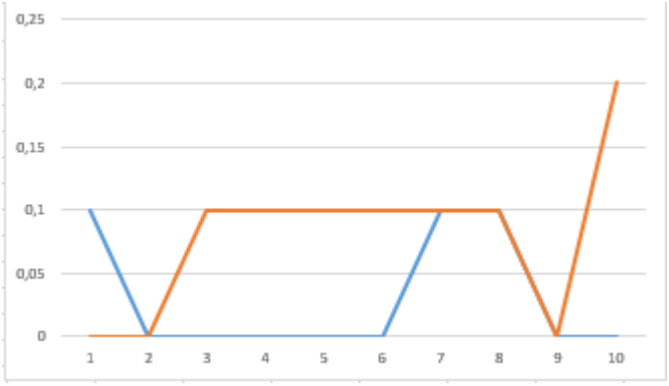

Рисунок 4.21 — Графік порівняння затраченого часу на видалення всіх документів до БД

З отриманого графіку можна сказати що операції видалення елементів на React-Native займає більше часу ніж на React.

Графік затраченого часу на сортування бульбашковим методом в Array наведено на рисунку 4.22:

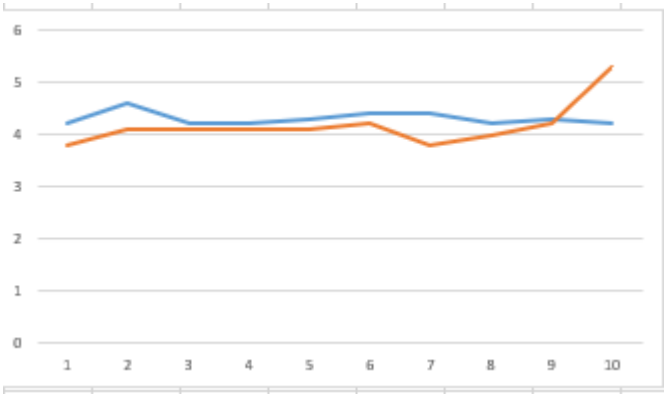

Рисунок 4.22 — Графік порівняння затраченого часу на сортування бульбашкою

З отриманого графіку можна сказати що операції сортування бульбашкою займає приблизно однаковий час, проте на React-Native – більше.

Графік затраченого часу на сортування методом злиття в Array наведено на рисунку 4.23.

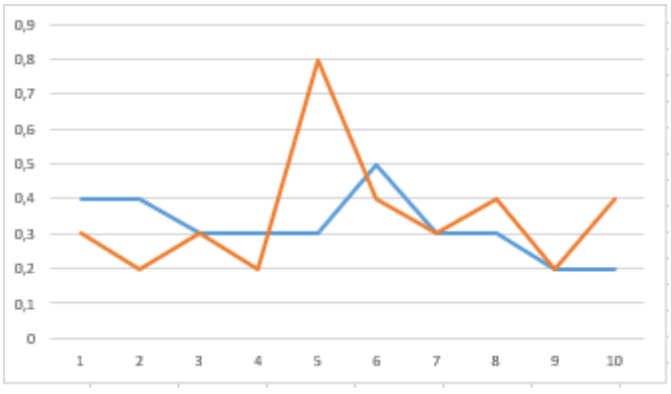

Рисунок 4.23 — Графік порівняння затраченого часу на сортуванням методом злиття

З отриманого графіку можна сказати що операції сортування елементів методом злиття займає приблизно однаковий час, проте на React-native займає менше.

Графік затраченої пам’яті на виконання всіх операцій наведено на рисунку 4.24:

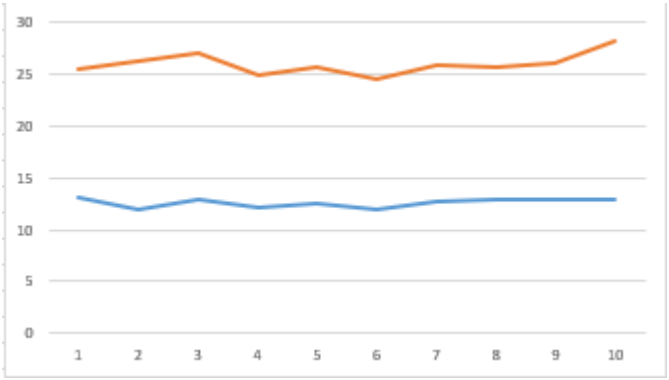

Рисунок 4.24 — Графік порівняння витраченої пам’яті

З отриманого графіку можна сказати що виконання операцій на React-Native використовує пам’яті більше ніж на React.

Графік затраченого часу на додавання в структуру даних List представлено на рисунку 4.25:

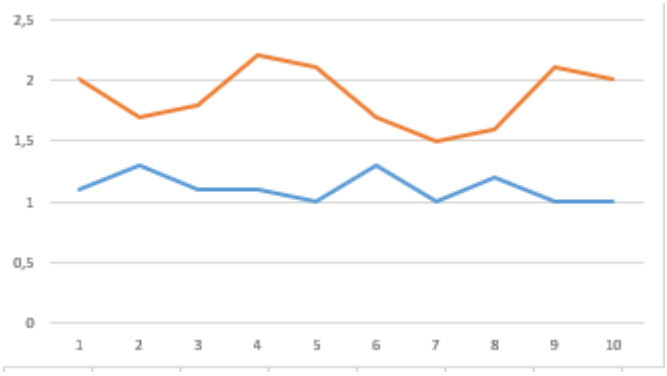

Рисунок 4.25 – Графік порівняння затраченого часу на додавання елементів

З отриманого графіку можна сказати що операції додавання елементів на React займає набагато менше часу ніж на React-Native.

Графік затраченого часу на пошук серед 500 елементів для структури даних List представлено на малюнку 4.26:

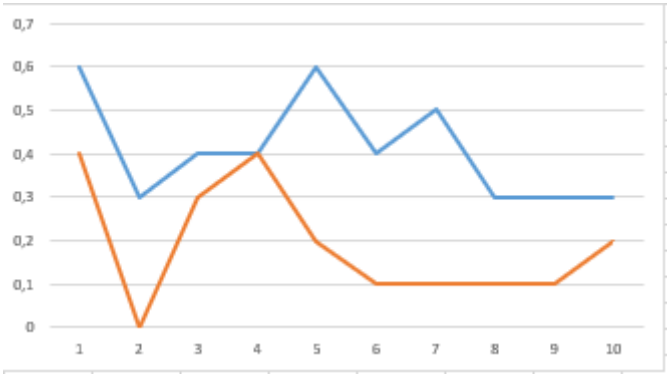

Рисунок 4.26 – Графік порівняння затраченого часу на пошук елементів

З отриманого графіку можна сказати що операції пошуку елементів на React займають більше ніж на React-Native.

Графік затраченого часу на видалення елементів для структури даних List наведено на рисунку 4.27:

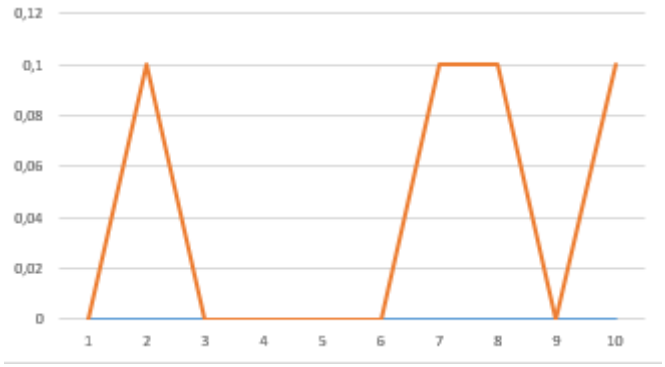

Рисунок 4.27 – Графік порівняння затраченого часу на видалення всіх елементів

З отриманого графіку можна сказати що операції видалення елементів на React-Native займає набагато більше часу ніж на React.

Графік затраченого часу на сортування бульбашковим методом в List наведено на рисунку 4.28:

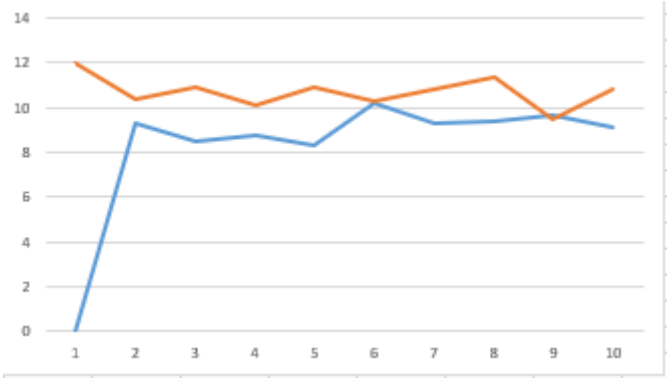

Рисунок 4.28 – Графік порівняння затраченого часу на сортування бульбашкою

З отриманого графіку можна сказати що операції сортування елементів бульбашкою займає приблизно однаковий час, проте на React – менше.

Графік затраченого часу на сортування методом злиття в Array наведено на рисунку 4.29:

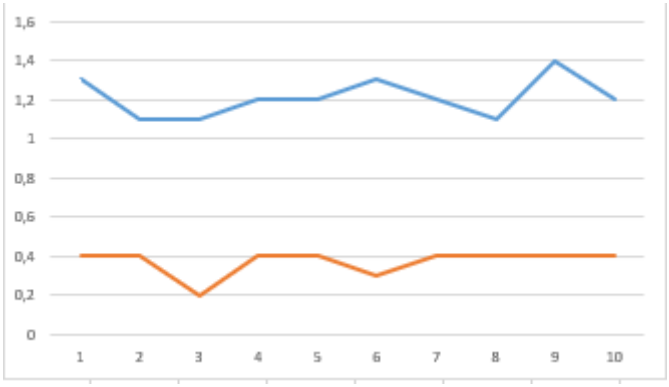

Рисунок 4.29 – Графік порівняння затраченого часу на сортуванням методом злиття

З отриманого графіку можна сказати що операції сортування елементів методом злиття на React займає більше часу в порівнянні з React-Native.

Графік порівняння витраченої пам’яті на виконання всіх операцій для List наведено на рисунку 4.30:

Рисунок 4.30 – Графік порівняння витраченої пам’яті

З отриманого графіку можна сказати що виконання операцій на React-Native використовує пам’яті ніж додаток на React.

Графік затраченого часу на додавання елементів в структуру даних Map представлено на малюнку 4.31:

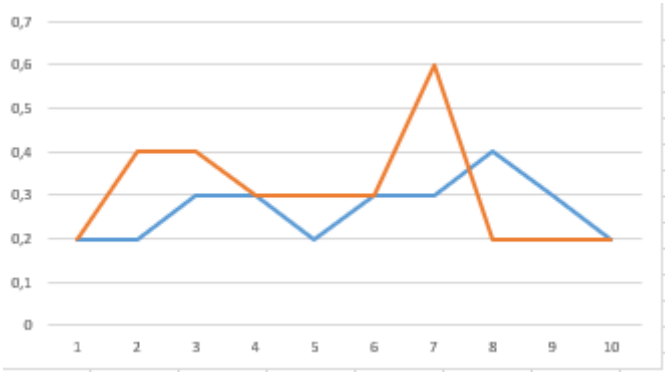

Рисунок 4.31 – Графік порівняння затраченого часу на додавання елементів

З отриманого графіку можна сказати що операції додавання елементів на React займає менше часу в порівнянні з React-Native.

Графік затраченого часу на пошук серед елементів для структури даних Map представлено на малюнку 4.32:

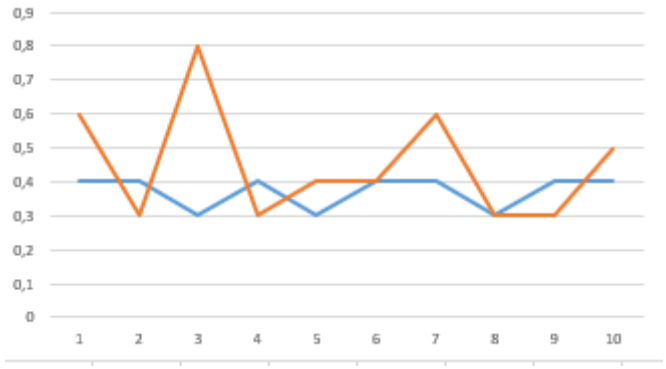

Рисунок 4.32 – Графік порівняння затраченого часу на пошук елементів

З отриманого графіку можна сказати що операції пошуку елементів на React займає менше часу в порівнянні з React-Native.

Графік затраченого часу на видалення елементів для структури даних Map наведено на рисунку 4.33:

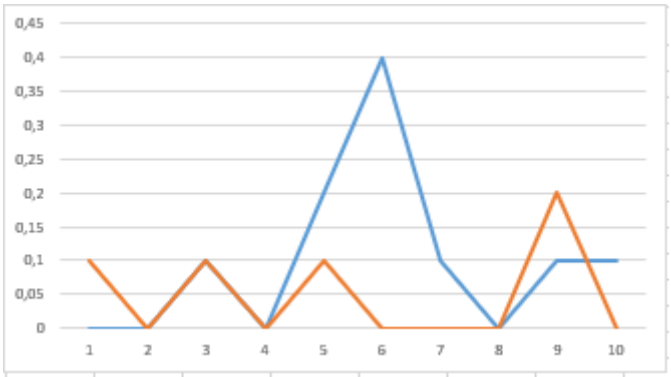

Рисунок 4.33 – Графік порівняння затраченого часу на видалення всіх елементів

З отриманого графіку можна сказати що операції видалення елементів на React займає більше часу в порівнянні з React-Native

Графік затраченого часу на сортування бульбашковим методом в Map наведено на рисунку 4.34:

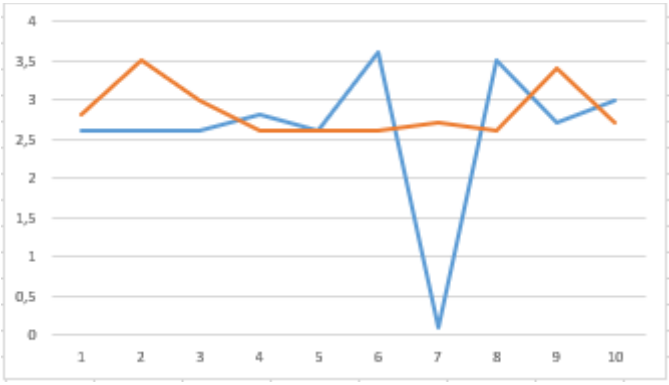

Рисунок 4.34 – Графік порівняння затраченого часу на сортування бульбашкою

З отриманого графіку можна сказати що операції сортування елементів бульбашкою на React займає більше часу в порівнянні з React-Native.

Графік затраченого часу на сортування методом злиття в Map наведено на рисунку 4.35:

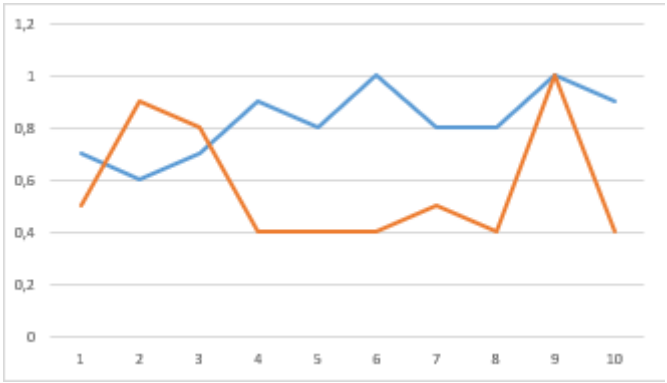

Рисунок 4.35 – Графік порівняння затраченого часу на сортуванням методом злиття

З отриманого графіку можна сказати що операції сортування елементів методом злиття на React займає більше часу в порівнянні з React-Native.

Графік порівняння витраченої пам’яті на виконання всіх операцій для Map наведено на рисунку 4.36:

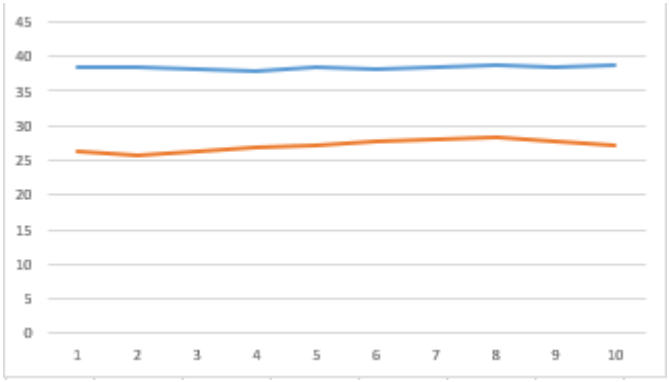

Рисунок 4.36 – Графік порівняння витраченої пам’яті

З отриманого графіку можна сказати що виконання операцій на React використовує пам’яті більше ніж на React

4.3 Дослідження роботи під час опрацювання 10000 елементів

Під час дослідження часу виконання операцій з обробки даних у розроблених додатках на основі React та React Native, було отримано результати вимірювань, які наведено в таблиці 4.25, 4.26, 4.27, 4.28, 4.29, 4.30 та таблиці 4.31, 4.32, 4.33, 4.34, 4.35, 4.36 :

Таблиця 4.25 – Час виконання операцій для React на основі Array

№	Add	Find	Delete	Bubble S	Merge S
1	194,2	3,1	0,3	507	3,50
2	184,7	3,1	0,3	509,2	3,8
3	195,9	3	0,4	529,9	4,2
4	184,2	2,8	0,29	508,3	4,50
5	185	3	0,4	523,2	4,1
6	191,8	3	0,4	510,4	3,2
7	189,5	2,8	0,3	499,6	4,3
8	193,9	3,3	0,3	533	4,4
9	191,3	2,7	0,29	490,8	4
10	183,7	2,8	0,20	510,7	4,2

Таблиця 4.26 – Кількість використаної пам’яті для React на основі Array

№	Memory

1	20,551
2	19,481
3	20,341
4	21,68
5	23,114
6	26,437
7	25,081
8	26,318
9	27,354
10	28,381

Таблиця 4.27 – Час виконання операцій для React на основі List

№	Add	Find	Delete	Bubble S	Merge S
1	104,5	2,8	0	1005,5	50,7
2	103,1	2,9	0,3	936,6	51,3
3	102,4	3	0,1	985,1	49,3
4	108,8	3,5	0,1	1070,3	50,6
5	103,5	2,8	0,2	950,3	47,6
6	107	3,3	0,1	972,4	49,9
7	104	2,8	0	978,8	49,5
8	98	2,7	0	995,1	48,8
9	99,7	2,8	0	973	48,2
10	100,6	3,2	0,3	964,6	51,7

Таблиця 4.28 – Кількість використаної пам’яті для React на основі List

№	Memory
1	28,756
2	30,122
3	31,84
4	33,611
5	33,106
6	36,211
7	35,33
8	39,119
9	40,096
10	39,329

Таблиця 4.29 – Час виконання операцій для React на основі Map

№	Add	Find	Delete	Bubble S	Merge S
1	3,6	4,4	0,7	312,5	5,9
2	3,6	3,4	0,8	324,4	6,3
3	3,2	3,2	1	312,7	6,5
4	3,1	3,1	1	321,4	6,4
5	3	2,9	0,8	320,8	6,8
6	3,2	3,1	0,8	303,9	6,3
7	3	3,1	0,9	304,5	6,5
8	3,2	3,1	0,9	314,2	7,6
9	4,8	3,7	1	323,4	6,1
10	3,2	3,5	0,7	317,3	6,9

Таблиця 4.30 – Кількість використаної пам’яті для React на основі Map

№	Memory
1	43,151
2	45,4
3	45,08
4	47,377
5	47,621
6	48,481
7	50,907
8	50,095
9	53,575
10	53,415

Таблиця 4.31 – Час виконання операцій для React-Native на основі Array

№	Add	Find	Delete	Bubble S	Merge S
1	203,7	0	0,3	532,9	9,2
2	199,7	0	0,2	506,7	3,6
3	193,1	0	0,3	513,4	3,2
4	194,6	0	0,3	512,2	3,4
5	195,4	0	0,3	499,8	3,1
6	188,2	0,1	0,2	490,9	3,4
7	193,9	0	0,3	505,7	3,1
8	33,953	0	0,3	493,9	3,2
9	185,1	0	0,3	472,1	3,7
10	193,9	0	0,2	495,9	4

Таблиця 4.32 – Кількість використаної пам’яті для React-Native на основі Array

№	Memory
1	27,033
2	35,186
3	33,843
4	33,656
5	33,871
6	33,868
7	33,953
8	33,944
9	33,948
10	33,567

Таблиця 4.33 – Час виконання операцій для React-Native на основі List

№	Add	Find	Delete	Bubble S	Merge S
1	112,6	1,1	0,1	974,9	2,7
2	125,2	1	0,2	933,8	3,4
3	132,6	1,1	0,3	924,3	3
4	127,1	1,1	0,3	943	3
5	118,6	1,1	0,2	932,4	3,3
6	125,8	1,1	0,2	959,9	2,3
7	126,1	1,1	0,3	920,5	2,7
8	129,8	1,1	0,3	918,1	2,7
9	123,7	1,1	0,3	971,6	2,4
10	135,7	1	0,3	932,2	2,6

Таблиця 4.34 – Кількість використаної пам’яті для React-Native на основі List

№	Memory
1	27,816
2	27,056
3	30,179
4	27,899
5	27,154
6	29,296
7	29,825
8	29,262
9	30,516
10	29,138

Таблиця 4.35 – Час виконання операцій для React-Native на основі Map

№	Add	Find	Delete	Bubble S	Merge S
1	2,7	3	0,7	318,3	4,8
2	2,9	2,9	1,1	320,8	7,2
3	2,8	3,3	0,8	325,2	4,8
4	3,4	3,3	0,8	333,5	6,9
5	2,6	3,1	0,8	315,7	5,3
6	3,5	3,5	1,1	347,7	6,9
7	2,6	3,3	0,7	321,1	5,2
8	3,9	3,4	1,4	342,5	6,6
9	2,7	3	1	303,5	4,5
10	3,5	3,4	1,2	351,5	8,4

Таблиця 4.36 – Кількість використаної пам’яті для React-Native на основі Map

№	Memory
1	30,15
2	32,64
3	36,59
4	33,2
5	37,54
6	35,67
7	39,98
8	40,39
9	42,92
10	41,76

З отриманих даних було складено порівняльні графіки.

Графік затраченого часу на додавання елементів в структуру даних Array представлено на малюнку 4.37:

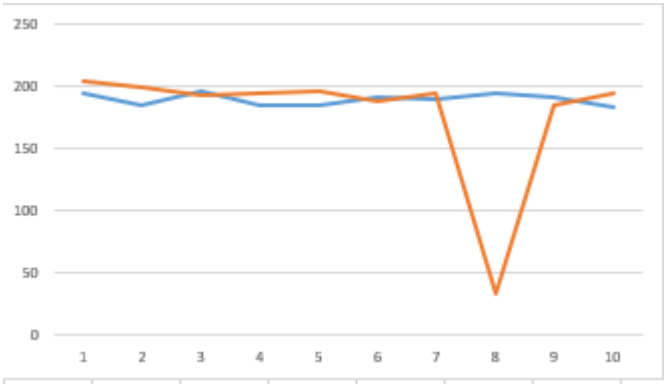

Рисунок 4.37 — Графік порівняння затраченого часу на додавання елементів

З отриманого графіку можна сказати що операції додавання елементів на React-Native займає іноді менше часу ніж додавання на React.

Графік затраченого часу на пошук серед елементів для структури даних Array представлено на малюнку 4.38:

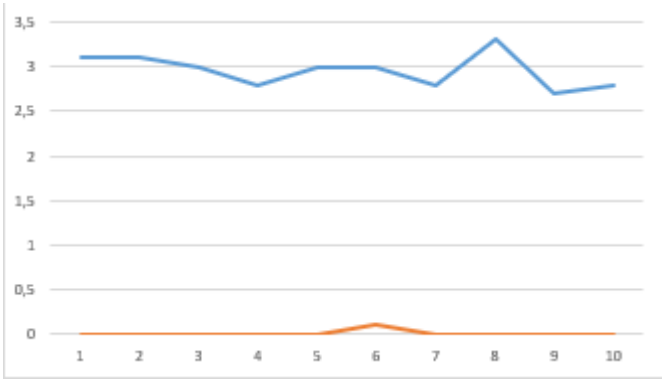

Рисунок 4.38 — Графік порівняння затраченого часу на пошук елементів

З отриманого графіку можна сказати що операції пошуку елементів на React займає більше часу ніж на React-Native.

Графік затраченого часу на видалення елементів для структури даних Array наведено на рисунку 4.39:

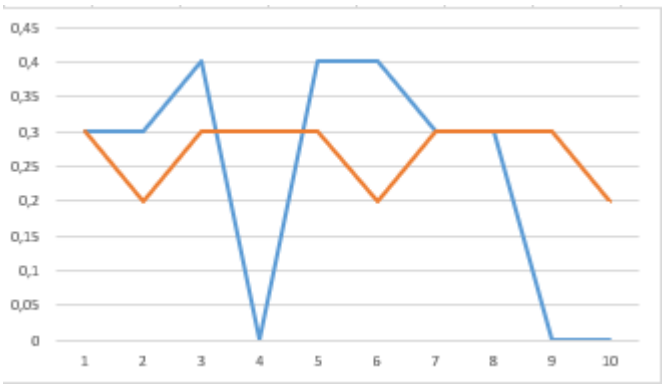

Рисунок 4.39 — Графік порівняння затраченого часу на видалення всіх елементів

З отриманого графіку можна сказати що операції видалення елементів займає приблизно однаковий час на обох фреймворках, проте React більше.

Графік затраченого часу на сортування бульбашковим методом в Array наведено на рисунку 4.40:

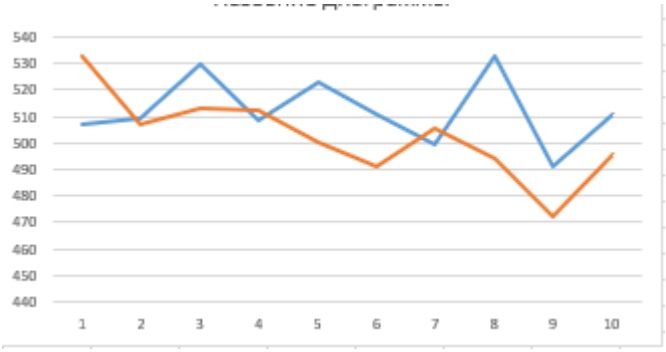

Рисунок 4.40 — Графік порівняння затраченого часу на сортування бульбашкою

З отриманого графіку можна сказати що операції сортування бульбашкою займає приблизно однаковий час, проте на React – більше.

Графік затраченого часу на сортування методом злиття в Array наведено на рисунку 4.41.

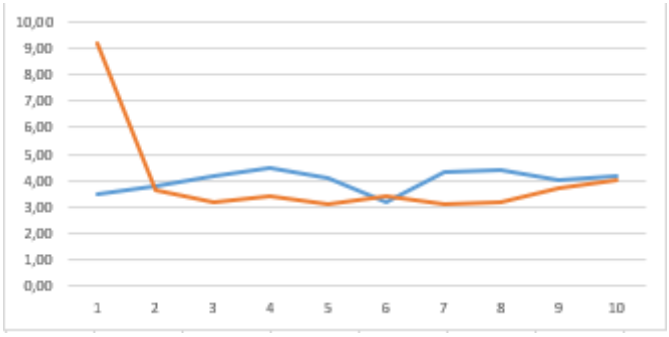

Рисунок 4.41 — Графік порівняння затраченого часу на сортуванням методом злиття

З отриманого графіку можна сказати що операції сортування елементів методом злиття займає приблизно однаковий час, графік симетричний, проте на React займає більше.

Графік затраченої пам’яті на виконання всіх операцій наведено на рисунку 4.42:

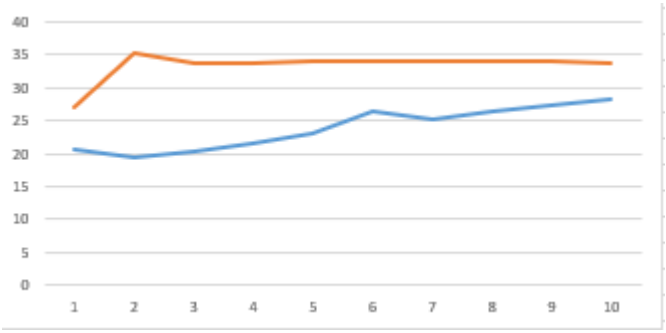

Рисунок 4.42 — Графік порівняння витраченої пам’яті

З отриманого графіку можна сказати що виконання операцій на React-Native використовує пам’яті більше ніж на React.

Графік затраченого часу на додавання елементів в структуру даних List представлено на рисунку 4.43:

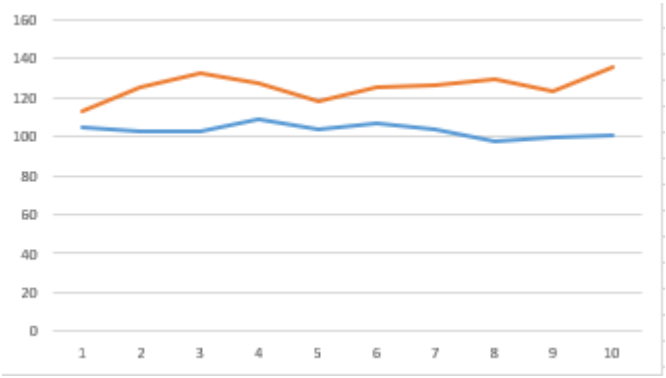

Рисунок 4.43 – Графік порівняння затраченого часу на додавання елементів

З отриманого графіку можна сказати що операції додавання елементів на React займає менше часу ніж на React-Native.

Графік затраченого часу на пошук серед елементів для структури даних List представлено на малюнку 4.44:

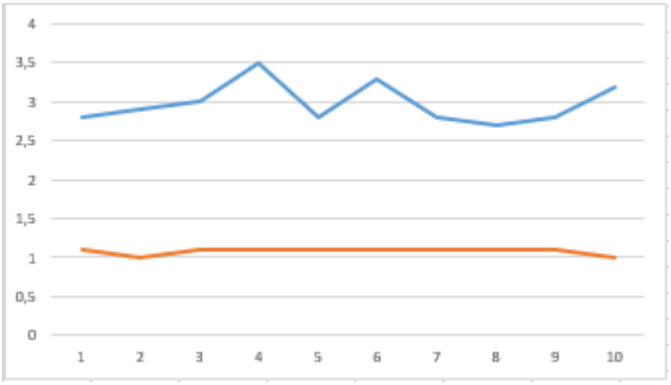

Рисунок 4.44 – Графік порівняння затраченого часу на пошук елементів

З отриманого графіку можна сказати що операції пошуку елементів на React займають більше часу ніж на React-Native.

Графік затраченого часу на видалення елементів для структури даних List наведено на рисунку 4.45:

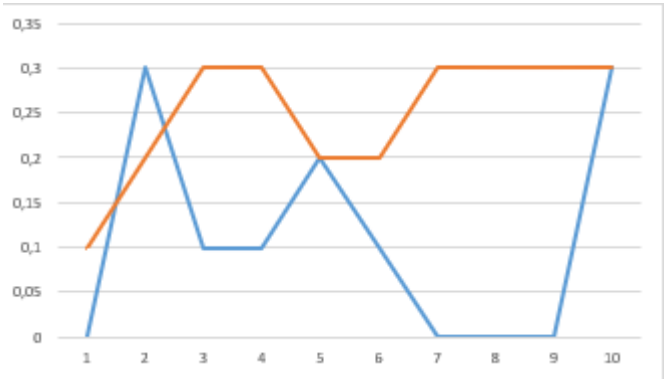

Рисунок 4.45 – Графік порівняння затраченого часу на видалення всіх елементів

З отриманого графіку можна сказати що операції видалення елементів на React-Native займає більше часу ніж на React

Графік затраченого часу на сортування бульбашковим методом в List наведено на рисунку 4.46:

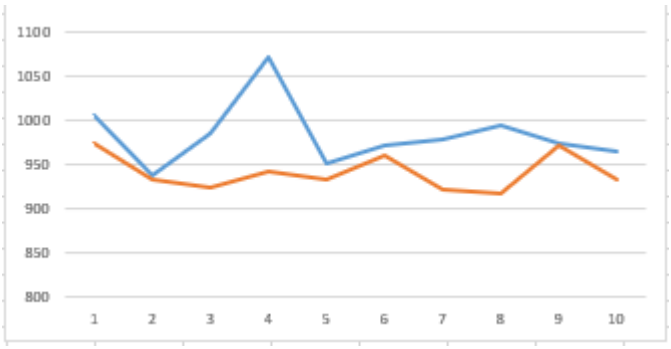

Рисунок 4.46 – Графік порівняння затраченого часу на сортування бульбашкою

З отриманого графіку можна сказати що операції сортування елементів бульбашкою займає приблизно однаковий час, однак на React – більше.

Графік затраченого часу на сортування методом злиття в List наведено на рисунку 4.47:

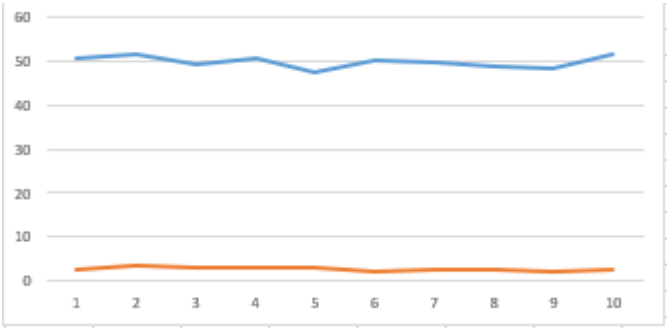

Рисунок 4.47 – Графік порівняння затраченого часу на сортуванням методом злиття

З отриманого графіку можна сказати що операції сортування елементів методом злиття на React займає набагато більше часу в порівнянні з React-Native

Графік порівняння витраченої пам’яті на виконання всіх операцій для List наведено на рисунку 4.48:

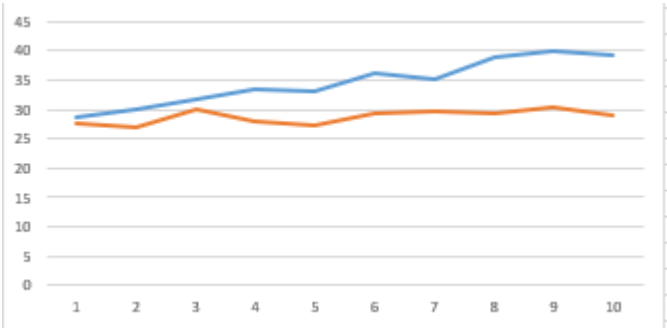

Рисунок 4.48 – Графік порівняння витраченої пам’яті

З отриманого графіку можна сказати що виконання операцій на React використовує пам’яті більше ніж на React-Native.

Графік затраченого часу на додавання елементів в структуру даних Map представлено на малюнку 4.49:

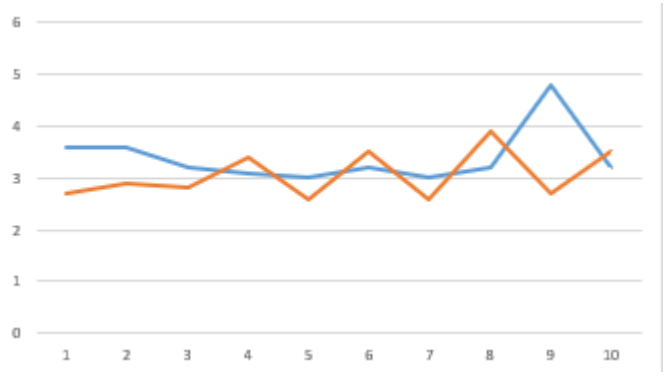

Рисунок 4.49 – Графік порівняння затраченого часу на додавання елементів

З отриманого графіку можна сказати що операції додавання елементів на React займає стільки же часу як і на React-Native.

Графік затраченого часу на пошук серед елементів для структури даних Map представлено на малюнку 4.50:

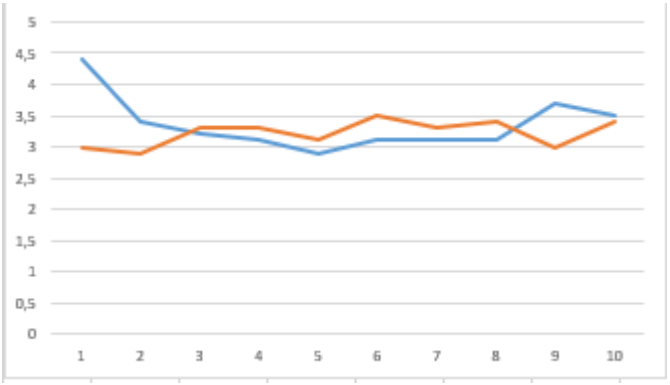

Рисунок 4.50 – Графік порівняння затраченого часу на пошук елементів

З отриманого графіку можна сказати що операції пошуку елементів на React займає стільки ж часу як і на React-Native.

Графік затраченого часу на видалення елементів для структури даних Map наведено на рисунку 4.51:

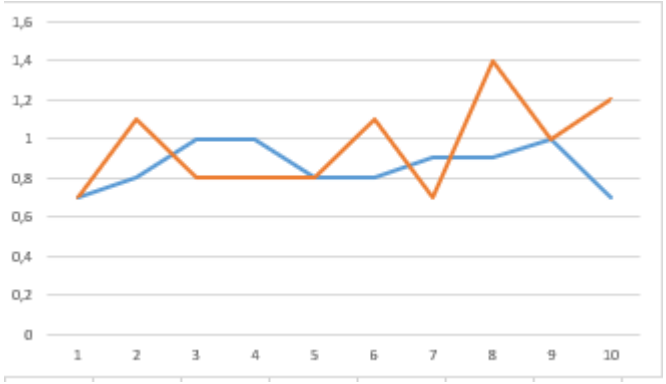

Рисунок 4.51 – Графік порівняння затраченого часу на видалення всіх елементів

З отриманого графіку можна сказати що операції видалення елементів на React займає менше часу в порівнянні з React-Native

Графік затраченого часу на сортування бульбашковим методом в Map наведено на рисунку 4.52:

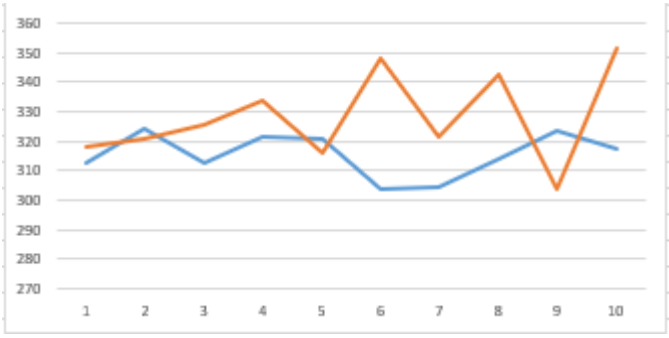

Рисунок 4.52 – Графік порівняння затраченого часу на сортування бульбашкою

З отриманого графіку можна сказати що операції сортування елементів бульбашкою на React займає менше часу в порівнянні з React-Native

Графік затраченого часу на сортування методом злиття в Map наведено на рисунку 4.53:

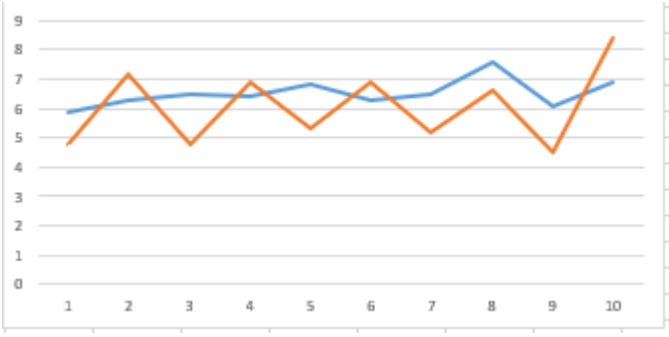

Рисунок 4.53 – Графік порівняння затраченого часу на сортуванням методом злиття

З отриманого графіку можна сказати що операції сортування елементів методом злиття на React займає приблизно стільки ж часу як і на React-Native.

Графік порівняння витраченої пам’яті на виконання всіх операцій для Map наведено на рисунку 4.54:

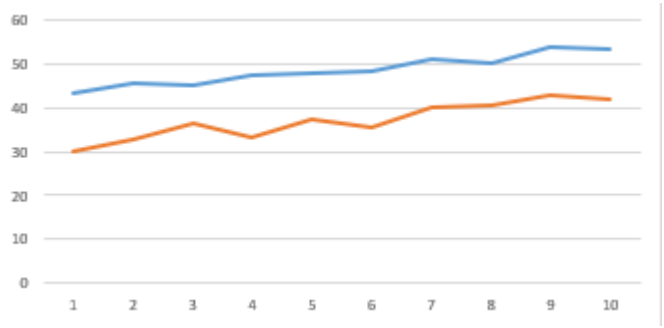

Рисунок 4.54 – Графік порівняння витраченої пам’яті

З отриманого графіку можна сказати що виконання операцій на React використовує пам’яті більше ніж на React.

4.2 Висновки за розділом 4

У ході проведення дослідження було проаналізовано продуктивність додатків, розроблених за допомогою React та React Native, для обробки структур даних Array, List та Map. Результати тестування, зокрема час виконання операцій (додавання, пошук, видалення, сортування) та використання пам’яті, демонструють наступні ключові аспекти:

Час виконання операцій:

- для більшості операцій додатки на React продемонстрували стабільні показники продуктивності з меншою затратою часу, особливо при обробці великих обсягів даних;
- React Native, хоча і забезпечив порівнянну продуктивність, мав більші затримки у виконанні складних операцій, таких як сортування та видалення елементів.

Використання пам’яті:

Додатки на React продемонстрували більшу оптимізацію використання пам’яті для масивів та списків, тоді як React Native показав більший обсяг використаної пам’яті при роботі з великими структурами даних.

Сортування:

Сортування методом злиття було ефективнішим у середовищі React, тоді як React Native продемонстрував трохи повільніший час виконання для обох алгоритмів сортування.

Загальні тенденції:

Для операцій додавання та пошуку React Native показав конкурентний час виконання, що вказує на потенційну ефективність цієї платформи для динамічних додатків із частими запитами.

Отримані результати підтверджують, що вибір технології має базуватись на конкретних вимогах проекту:

- React підходить для складних завдань із великими обсягами даних;
- React Native є доцільним вибором для додатків, орієнтованих на мобільні платформи, з менш жорсткими вимогами до часу виконання операцій.

ЗАГАЛЬНИЙ ВИСНОВОК

У процесі виконання дипломної роботи було проведено аналіз ресурсоємності веб-додатків, створених за допомогою React і React Native. Було розглянуто основні поняття та принципи розробки веб-додатків, а також особливості архітектури й функціональності обох платформ.

У ході роботи було створено два тестових додатки, які дозволяли виконувати операції з різними структурами даних (масиви, списки, мапи). Проведено тестування додатків із вимірюванням часу виконання операцій і споживанням пам’яті для таких операцій:

- додавання елементів;
- пошук елементів;
- видалення елементів;
- сортування даних (алгоритми bubble sort і merge sort).

Проведено порівняльний аналіз отриманих результатів, який дозволив встановити переваги та недоліки використання React і React Native залежно від умов і завдань:

- React показав високу ефективність і стабільність для роботи з великими обсягами даних;
- React Native, хоча і поступається в продуктивності, є більш підходящим для мобільних платформ і кросплатформних додатків.

Окрему увагу було приділено використанню бібліотек для автоматизації процесів і оптимізації роботи додатків, таких як react-hook-form і faker.js, що спрощують генерацію даних і тестування.

Загалом, робота підтвердила важливість ретельного вибору технологій для забезпечення ресурсоефективності додатків, а також запропонувала рекомендації щодо використання React і React Native залежно від специфіки проєкту.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. **React Documentation**. (2024). Доступно на: https://reactjs.org/docs/getting-started.html.
2. **React Native Documentation**. (2024). Доступно на: https://reactnative.dev/docs/getting-started.
3. Fowler, M. (2004). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional.
4. McDowell, G. L. (2015). *Cracking the Coding Interview: 189 Programming Questions and Solutions*. CareerCup.
5. Седжвік, Р., Вейн, К. (2011). *Алгоритми: побудова та аналіз*. Київ: Видавництво КМА.
6. **Mozilla Developer Network (MDN)**. (2024). *JavaScript Performance API*. Доступно на: https://developer.mozilla.org/en-US/docs/Web/API/Performance.
7. Wróbel, P. (2023). *Hands-On Data Structures and Algorithms with JavaScript*. Packt Publishing.
8. Дейвіс, Д. (2020). *Практичний посібник з оптимізації JavaScript*. Видавництво JavaScript Masters.
9. Маннінг, К. (2021). *React Performance Optimization Techniques*. Оксфорд: Маннінг Паблішинг.
10. **GitHub Open Source Projects**. (2024). Доступно на: https://github.com.
11. Браун, Е. (2019). *JavaScript: The Good Parts*. O'Reilly Media.

ДОДАТОК А

Технічне завдання

ЗАТВЕРДЖУЮ
Проректор Українського державного
університету науки і технологій
Анатолій РАДКЕВИЧ

«АНАЛІЗ РЕСУРСОЄМНОСТІ ВЕБ-ДОДАТКІВ РОЗРОБЛЕНИХ ЗА ДОПОМОГОЮ REACT ТА REACT-NATIVE»
Технічне завдання
44165850.1434-01

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН
Керівник розробки
_____Олександр ІВАНОВ
Виконавець
_____Кирило ДОРОГОКУПЛЯ
Нормоконтролер
_____Світлана ВОЛКОВА

2025

ЗАТВЕРДЖЕНО
44165850.1434-01

«АНАЛІЗ РЕСУРСОЄМНОСТІ ВЕБ-ДОДАТКІВ РОЗРОБЛЕНИХ ЗА ДОПОМОГОЮ REACT ТА REACT-NATIVE»

Технічне завдання
44165850.1434-01
Листів 15

2025

44165850.1434-01
ЗМІСТ

ВСТУП

В сучасному світі інформаційних технологій все більшої актуальності набуває питання оптимізації ресурсоємності веб-додатків. Веб-додатки, розроблені з використанням сучасних JavaScript-бібліотек, таких як React та React Native, стають основними інструментами для створення інтерактивних інтерфейсів та кросплатформних рішень.

React та React Native забезпечують високий рівень продуктивності та зручності розробки завдяки використанню компонентного підходу та віртуального DOM. Проте їх використання породжує питання ефективності використання апаратних ресурсів, особливо в умовах роботи з великими обсягами даних та виконанням складних операцій.

Мета цього дослідження — аналіз ресурсоємності веб-додатків, розроблених на основі React та React Native, а також встановлення їхніх переваг і недоліків у контексті продуктивності.

У рамках роботи проводиться:

- порівняння часу виконання ключових операцій (додавання, видалення, пошуку, сортування) з різними обсягами даних (500, 1000, 10000 елементів);
- аналіз використання пам’яті;
- виявлення залежності продуктивності від архітектури додатків.

Основна увага приділяється практичному застосуванню результатів дослідження, що дозволяє не лише отримати теоретичні знання про можливості React і React Native, але й визначити оптимальні підходи до розробки ефективних веб-додатків.

Таким чином, результати цього дослідження спрямовані на надання практичних рекомендацій щодо вибору технології залежно від вимог до продуктивності та конкретних умов використання.

44165850.1434-01

1 ПІДСТАВА ДЛЯ РОЗРОБКИ

Основою для розробки є наказ ректора Українського державного університету науки і технології Радкевич А.В. «Про затвердження тем та призначення керівників дипломних проектів» №1186 ст від 29.12. 2023 року.

Тема проекту: «Аналіз ресурсоємності веб-додатків розроблених за допомогою React та React-native».

Керівник дипломного проекту: Іванов О. П.

2 ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою розробки є створення інструментальних засобів для аналізу ресурсоємності веб-додатків, побудованих за допомогою React та React Native, що дозволить оцінити ефективність і продуктивність цих технологій. Дослідження спрямоване на розв’язання таких задач:

– вимірювання часу виконання базових операцій (додавання, пошук, видалення, сортування) на різних структурах даних;

– аналіз використання оперативної пам’яті під час виконання операцій;

– порівняння продуктивності додатків, розроблених на обох платформах, для різних обсягів даних.

Розробка призначена для:

– надання рекомендацій розробникам щодо вибору відповідної технології залежно від вимог проекту;

– вивчення впливу архітектурних особливостей React і React Native на продуктивність веб-додатків

– створення основи для подальшого вдосконалення процесів розробки кросплатформних рішень.

Дана система буде корисною для студентів, дослідників і розробників, які працюють над оптимізацією ресурсів веб-додатків або вибором відповідних інструментів для вирішення конкретних задач.

2.1 Функціональне призначення розробки

Аналіз продуктивності веб-додатків:

– Основною функцією розробки є детальне дослідження ефективності та продуктивності веб-додатків, створених за допомогою фреймворків React і React Native. Це включає вивчення ресурсів, необхідних для виконання операцій додавання, пошуку, видалення, сортування даних та інших базових функцій.

Порівняння продуктивності:

– Розробка дозволяє провести порівняльний аналіз обох платформ для виявлення переваг та недоліків їх застосування у різних сценаріях. Зокрема, оцінюється час виконання операцій та обсяг використаної пам’яті.

Практичне впровадження:

– Вивчення архітектурних особливостей та функціональності обох фреймворків надає можливість визначити оптимальні практичні застосування для кожного з них. Це може включати створення кросплатформних додатків, оптимізацію процесів роботи з даними, адаптацію до вимог різних проєктів.

Рекомендації розробникам:

– Результати дослідження спрямовані на формування рекомендацій для розробників щодо вибору фреймворку залежно від вимог конкретного проєкту, враховуючи продуктивність і ресурсні обмеження.

Документація та висновки:

– Завершення роботи передбачає підготовку документації, яка включає результати аналізу, порівняльні діаграми продуктивності, рекомендації для використання та висновки для подальшого вдосконалення розробки.

2.2 Експлуатаційне призначення розробки:

Оцінка та вибір фреймворку:

Результати аналізу допоможуть організаціям і розробникам визначити, який із фреймворків — React чи React Native — найкраще підходить для конкретних потреб проєкту. Це стосується як веб-додатків, так і мобільних рішень.

Планування розробки проєктів:

Отримані знання про продуктивність фреймворків дозволять розробникам ефективніше планувати та реалізовувати проєкти, які потребують високої продуктивності та оптимального використання ресурсів.

Оптимізація використання ресурсів:

Глибокий аналіз продуктивності операцій із масивами даних і списками допоможе оптимізувати обробку даних та управління пам’яттю під час розробки додатків.

Підтримка та вдосконалення додатків:

Використання отриманих результатів забезпечить можливість підтримки та розвитку існуючих проєктів, враховуючи потреби користувачів у масштабованих і ресурсоефективних рішеннях.

Документація та рекомендації:

Підготовлена документація включатиме кращі практики, результати порівняльного аналізу, а також рекомендації для вибору фреймворку залежно від типу застосування.

Підсумок:

Експлуатаційне призначення розробки спрямоване на практичне використання результатів дослідження для покращення якості програмного забезпечення, яке розробляється із використанням React і React Native, а також на впровадження рекомендацій для вибору оптимального інструменту для конкретного проєкту.

3 ВИМОГИ ДО ПРОГРАМИ

3.1 Вимоги до функціональних характеристик

Основні функціональні вимоги до аналізу продуктивності веб-додатків на React та React Native:

Вимірювання продуктивності операцій:

– додавання даних: оцінка швидкості додавання елементів у структури даних;

– видалення даних: час, необхідний для видалення кожного другого елемента з колекцій;

– пошук даних: оцінка швидкості пошуку елементів у перших, середніх і останніх сегментах масиву;

– сортування даних: продуктивність алгоритмів сортування, таких як Bubble Sort та Merge Sort.

Вимірювання використання пам’яті:

– кількість оперативної пам’яті, яку споживає додаток для кожної з операцій.

Масштабованість:

– продуктивність та стабільність роботи додатків при збільшенні обсягів даних (500, 1000, 10000 елементів).

Робота з великими масивами:

– Оцінка ефективності обробки структур даних великого розміру.

Ці вимоги до функціональних характеристик дослідження веб-додатків, розроблених на основі React і React Native, допоможуть оцінити їхню продуктивність, ефективність та обрати найбільш відповідний інструмент для реалізації проектів залежно від конкретних потреб і задач.

Вхідні дані:

Структура даних:

– опис властивостей елементів у масивах чи списках: унікальний ідентифікатор (id), ім’я (name), адреса (address) та інші поля.

Кількість даних:

– масиви даних із 500, 1000, і 10,000 елементів.

Алгоритми:

– Bubble Sort, Merge Sort для сортування елементів.

Сценарії використання:

– виконання CRUD-операцій: додавання, пошук, видалення та сортування елементів.

Вихідні дані:

Результати аналізу продуктивності:

– таблиці з часом виконання кожної операції для різних обсягів даних.

Рекомендації для вибору інструментів:

– висновки про те, який із фреймворків є більш ефективним залежно від обсягу даних та типу операцій.

3.2 Вимоги до надійності

Вимоги до надійності веб-додатків на основі React і React Native такі:

– забезпечення повідомлень про стан роботи програми: під час виконання основних операцій, таких як додавання, видалення чи оновлення даних, додаток має виводити користувачеві повідомлення про успішність або помилки. Це дозволяє уникнути неоднозначності в роботі програми;

– створення резервної копії проєкту: наявність резервних копій вихідного коду на зовнішніх носіях або в хмарних сховищах для збереження змін у разі втрати локальної версії;

– забезпечення стабільності роботи: обробка можливих винятків, таких як відсутність доступу до серверів або помилки при обробці даних, з метою уникнення аварійного завершення роботи додатка;

– контроль споживання ресурсів: реалізація механізмів моніторингу та оптимізації використання пам’яті для уникнення витоків і перевантаження системи.

3.3 Вимоги експлуатації

Для роботи з програмою користувач повинен мати базові навички роботи з комп'ютером і веб-додатками. Основні вимоги до експлуатації:

– доступ до інструкції користувача: розробка супровідної документації, яка містить опис функціоналу додатка та порядок роботи з ним;

– вимоги до платформи: додаток повинен бути сумісний з сучасними браузерами для веб-версії та з мобільними пристроями для додатків React Native;

– навчання користувачів: користувач повинен ознайомитися з інтерфейсом додатка, який має бути інтуїтивно зрозумілим для ефективної взаємодії.

3.4 Вимоги до складу та параметрів технічних засобів

Продукти, що розробляється повинен використовуватись на пристроях, що мають наступні характеристики:

– операційна система: комп'ютер повинен працювати під управлінням операційної системи, яка підтримує виконання програмного продукту. Для прикладу, це може бути операційна система Windows, macOS або Linux;

– діагональ монітора: монітор комп'ютера повинен мати достатньою діагональ для зручного відображення інтерфейсу продукту та взаємодії з ним. Ідеально, це може бути монітор із середньою діагоналлю, наприклад, 21-24 дюйми;

– роздільна здатність монітора: монітор повинен мати достатньо високу роздільну здатність для чіткого та якісного відображення інтерфейсу продукту. Рекомендована роздільна здатність - Full HD (1920x1080) або вища;

– оперативна пам'ять (RAM): комп'ютер повинен мати достатньо оперативної пам'яті для ефективної роботи програмного продукту. Рекомендована кількість оперативної пам'яті - не менше 4 ГБ;

– вбудована пам'ять (жорсткий диск або SSD): комп'ютер повинен мати достатньо внутрішньої пам'яті для зберігання програмного продукту та його даних. Рекомендована ємність вбудованої пам'яті - не менше 256 ГБ;

– частота процесора: комп'ютер повинен мати процесор з достатньою частотою для швидкої обробки даних та виконання завдань програмного продукту. Рекомендована частота процесора - не менше 2.5 ГГц;

– порти та комунікації: комп'ютер повинен мати необхідні порти та комунікаційні можливості для підключення до мережі, зовнішніх пристроїв та інтернету. Доцільно мати USB-порти, Ethernet-порт для мережевого підключення, а також можливість підключення до Wi-Fi.

3.5 Вимоги до інформаційної та програмної сумісності

Програмні продукти розробляється для всіх видів операційних систем сімейства “Windows” починаючи від версії 10 та наступні версії.

4 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу документації мають входити:

- текст програми;
- керівництво користувача для користування мобільним додатком.

Вся документація програмних додатків повинна задовольняти вимоги до програмної документації.

44165850.1434-01

5 СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

Таблиця 5.1 – Стадії та етапи розробки

Стадія	Зміст	Строки виконання
Технічне завдання	Постановка задачі, збір інформації, виріб та обґрунтування критеріїв розробки. Попередній вибір методів рішення задач. Визначення вимог до технічних засобів. Узгодження і затвердження технічного завдання.	04.09.23 –15.09.23
Робочий проект	Програмування та відлагодження програми.	18.09.23 – 22.09.23
	Тестування програми	25.09.23 – 27.09.23
	Розробка, узгодження і затвердження програмної документації.	28.09.23 – 29.09.23

6 ПОРЯДОК ПРИЙНЯТТЯ

Контроль за виконанням роботи здійснює керівник розробки доц. Іванов О. П.

7 ТЕХНІЧНО-ЕКОНОМІЧНІ ПОКАЗНИКИ

Показники та їх розрахунок не були описані у документах бо розробка програмного продукту несе в собі навчальний характер, а не комерційний.

ДОДАТОК Б

Текст програми

ЗАТВЕРДЖУЮ
Проректор Українського державного
університету науки і технологій
Анатолій РАДКЕВИЧ

«АНАЛІЗ РЕСУРСОЄМНОСТІ ВЕБ-ДОДАТКІВ РОЗРОБЛЕНИХ ЗА ДОПОМОГОЮ REACT ТА REACT-NATIVE»
Текст програми
44165850.01434–01 12–01

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН
Керівник розробки
_____Олександр ІВАНОВ
Виконавець
_____Кирило ДОРОГОКУПЛЯ
Нормоконтролер
_____Світлана ВОЛКОВА

2025

ЗАТВЕРДЖЕНО

44165850.01434–01 12–01

АНАЛІЗ РЕСУРСОЄМНОСТІ ВЕБ-ДОДАТКІВ РОЗРОБЛЕНИХ ЗА ДОПОМОГОЮ REACT ТА REACT-NATIVE

Текст програми
44165850.01434–01 12–01
Листів 20

2025

44165850.01434–01 12

АНОТАЦІЯ

Документ 44165850.01434–01 12 01 «Аналіз ресурсоємності веб-додатків розроблених за допомогою React та React-native» входить до складу програмної документації на програму, що описує результати аналізу продуктивності та ефективності веб-додатків.

У даному документі представлений текст додатку. Додаток написаний на мові JavaScript з використанням фреймворків React та React-native у програмному середовищі Visual Studio Code.

ЗМІСТ

44165850.01434–01 12

1 ТЕКСТ ПРОГРАМИ

1.1 Текст файлу App.js React

```
// App.js
import React from 'react';
import { BrowserRouter as Router, Route, Routes, Link } from 'react-router-dom';
import ArrayComponent from './components/arrayComponent/ArrayComponent';
import ListComponent from './components/listComponent/ListComponent';
import MapComponent from './components/mapComponent/MapComponent';

function App() {
 return (
   <Router>
     <div>
       <nav>
         <ul>
           <li><Link to="/array">Array Operations</Link></li>
           <li><Link to="/list">List Operations</Link></li>
           <li><Link to="/map">Map Operations</Link></li>
         </ul>
       </nav>

       <Routes>
         <Route path="/array" element={<ArrayComponent />} />
         <Route path="/list" element={<ListComponent />} />
         <Route path="/map" element={<MapComponent />} />
       </Routes>
     </div>
   </Router>
 );
}

export default App;
```

1.2 Текст файлу ArrayComponent.js на React

```
import React, { useState } from 'react';
import { generateData } from '../../utils/dataGenerator'; // Импортируем функцию генерации данных

function ArrayComponent() {
 const [results, setResults] = useState([]);

 // Функция для измерения времени выполнения
 const measureExecutionTime = (func, ...args) => {
   const start = performance.now();
   func(...args);
   const end = performance.now();
   return end - start;
 };

 // Функция для измерения использования памяти
 const measureMemoryUsage = () => {
   if (performance.memory) {
     return performance.memory.usedJSHeapSize / (1024 * 1024); // в MB
   } else {
     return 'Memory API не поддерживается';
   }
 };

 // Удаление каждого второго элемента
 const deleteEverySecondElement = (arr) => {
   return arr.filter((_, index) => index % 2 === 0);
 };

 // Поиск первой и последней трети элементов
 const searchFirstAndLastThird = (arr) => {
   const oneThird = Math.floor(arr.length / 3);
```

```
  const firstThird = arr.slice(0, oneThird); // Первая треть
  const lastThird = arr.slice(-oneThird); // Последняя треть
  return { firstThird, lastThird };
};

// Bubble Sort
const bubbleSort = (arr) => {
  let n = arr.length;
  let swapped;
  do {
    swapped = false;
    for (let i = 0; i < n - 1; i++) {
      if (arr[i].id > arr[i + 1].id) {
        [arr[i], arr[i + 1]] = [arr[i + 1], arr[i]];
        swapped = true;
      }
    }
    n--;
  } while (swapped);
  return arr;
};

// Merge Sort
const mergeSort = (arr) => {
  if (arr.length <= 1) return arr;

  const mid = Math.floor(arr.length / 2);
  const left = mergeSort(arr.slice(0, mid));
  const right = mergeSort(arr.slice(mid));

  return merge(left, right);
};

const merge = (left, right) => {
  let result = [];
  let leftIndex = 0;
  let rightIndex = 0;

  while (leftIndex < left.length && rightIndex < right.length) {
    if (left[leftIndex].id < right[rightIndex].id) {
      result.push(left[leftIndex]);
      leftIndex++;
    } else {
      result.push(right[rightIndex]);
      rightIndex++;
    }
  }

  return result.concat(left.slice(leftIndex)).concat(right.slice(rightIndex));
};

// Базовые операции
const runOperations = (size) => {
  let array = generateData(size); // Генерируем данные с использованием Faker.js
  const times = [];
  const memoryUsage = [];

  // Добавление элементов
  const addTime = measureExecutionTime(() => {
    array = generateData(size); // Заполнение с использованием Faker.js
  });
  times.push({ operation: 'Добавление', time: addTime });

  // Поиск первой и последней трети
  const searchTime = measureExecutionTime(() => {
    const { firstThird, lastThird } = searchFirstAndLastThird(array);
    console.log('Первая треть:', firstThird);
    console.log('Последняя треть:', lastThird);
  });
  times.push({ operation: 'Поиск', time: searchTime });

  // Удаление каждого второго элемента
  const deleteTime = measureExecutionTime(() => {
    array = deleteEverySecondElement(array);
  });
```

```
  times.push({ operation: 'Удаление каждого второго элемента', time: deleteTime });

  // Bubble Sort
  const bubbleSortTime = measureExecutionTime(() => {
    bubbleSort([...array]); // Копируем массив для сортировки
  });
  times.push({ operation: 'Bubble Sort', time: bubbleSortTime });

  // Merge Sort
  const mergeSortTime = measureExecutionTime(() => {
    mergeSort([...array]); // Копируем массив для сортировки
  });
  times.push({ operation: 'Merge Sort', time: mergeSortTime });

  // Измерение использования памяти после операций
  memoryUsage.push(measureMemoryUsage());

  setResults({ times, memoryUsage });
 };

 return (
  <div>
   <h2>Операции с массивом</h2>
   <button onClick={() => runOperations(500)}>Запустить на 500 элементах</button>
   <button onClick={() => runOperations(1000)}>Запустить на 1000 элементах</button>
   <button onClick={() => runOperations(10000)}>Запустить на 10 000 элементах</button>

   <h3>Результаты:</h3>
   {results.times && (
    <ul>
     {results.times.map((result, index) => (
      <li key={index}>
       Операция: {result.operation}, Время выполнения: {result.time} мс
      </li>
     ))}
    </ul>
   )}
   {results.memoryUsage && (
    <p>Использование памяти: {results.memoryUsage} MB</p>
   )}
  </div>
 );
}

export default ArrayComponent;
```

1.3 Текст файлу ListComponent.js на React

```
import React, { useState } from 'react';
import { generateData } from '../../utils/dataGenerator'; // Импортируем генератор данных

class ListNode {
  constructor(value, next = null) {
    this.value = value;
    this.next = next;
  }
}

class LinkedList {
  constructor() {
    this.head = null;
  }

  add(value) {
    const newNode = new ListNode(value);
    if (!this.head) {
      this.head = newNode;
    } else {
      let current = this.head;
      while (current.next) {
```

```
      current = current.next;
    }
    current.next = newNode;
  }
}

searchFirstAndLastThird() {
  const arr = this.getArrayFromList();
  const oneThird = Math.floor(arr.length / 3);
  const firstThird = arr.slice(0, oneThird);
  const lastThird = arr.slice(-oneThird);
  return { firstThird, lastThird };
}

deleteEverySecondElement() {
  let current = this.head;
  let index = 0;

  while (current && current.next) {
    if (index % 2 === 0) {
      current.next = current.next.next;
    }
    current = current.next;
    index++;
  }
}

bubbleSort() {
  let arr = this.getArrayFromList();
  let n = arr.length;
  let swapped;

  do {
    swapped = false;
    for (let i = 0; i < n - 1; i++) {
      if (arr[i].id > arr[i + 1].id) {
        [arr[i], arr[i + 1]] = [arr[i + 1], arr[i]];
        swapped = true;
      }
    }
    n--;
  } while (swapped);

  this.fromArrayToList(arr);
}

mergeSort() {
  let arr = this.getArrayFromList();

  const merge = (left, right) => {
    let result = [];
    let leftIndex = 0;
    let rightIndex = 0;

    while (leftIndex < left.length && rightIndex < right.length) {
      if (left[leftIndex].id < right[rightIndex].id) {
        result.push(left[leftIndex]);
        leftIndex++;
      } else {
        result.push(right[rightIndex]);
        rightIndex++;
      }
    }
    return result.concat(left.slice(leftIndex)).concat(right.slice(rightIndex));
  };

  const mergeSortRec = (arr) => {
    if (arr.length <= 1) return arr;

    const mid = Math.floor(arr.length / 2);
    const left = mergeSortRec(arr.slice(0, mid));
    const right = mergeSortRec(arr.slice(mid));

    return merge(left, right);
  };
```

```
    arr = mergeSortRec(arr);
    this.fromArrayToList(arr);
  }

  getArrayFromList() {
    const arr = [];
    let current = this.head;
    while (current) {
      arr.push(current.value);
      current = current.next;
    }
    return arr;
  }

  fromArrayToList(arr) {
    this.head = null;
    for (let i = 0; i < arr.length; i++) {
      this.add(arr[i]);
    }
  }
}

function ListComponent() {
  const [results, setResults] = useState([]);

  const measureExecutionTime = (func, ...args) => {
    const start = performance.now();
    func(...args);
    const end = performance.now();
    return end - start;
  };

  const measureMemoryUsage = () => {
    if (performance.memory) {
      return performance.memory.usedJSHeapSize / (1024 * 1024); // в MB
    } else {
      return 'Memory API не підтримується';
    }
  };

  const runOperations = (size) => {
    const list = new LinkedList();
    const data = generateData(size); // Генерация данных
    const times = [];
    const memoryUsage = [];

    // Додавання елементів
    const addTime = measureExecutionTime(() => {
      data.forEach((item) => list.add(item));
    });
    times.push({ operation: 'Додавання', time: addTime });

    // Пошук першої і останньої третини
    const searchTime = measureExecutionTime(() => {
      const { firstThird, lastThird } = list.searchFirstAndLastThird();
      console.log('Перша третина:', firstThird);
      console.log('Остання третина:', lastThird);
    });
    times.push({ operation: 'Пошук', time: searchTime });

    // Видалення кожного другого елемента
    const deleteTime = measureExecutionTime(() => {
      list.deleteEverySecondElement();
    });
    times.push({ operation: 'Видалення кожного другого елемента', time: deleteTime });

    // Bubble Sort
    const bubbleSortTime = measureExecutionTime(() => {
      list.bubbleSort();
    });
    times.push({ operation: 'Bubble Sort', time: bubbleSortTime });

    // Merge Sort
    const mergeSortTime = measureExecutionTime(() => {
```

```
            list.mergeSort();
          });
          times.push({ operation: 'Merge Sort', time: mergeSortTime });

          memoryUsage.push(measureMemoryUsage());

          setResults({ times, memoryUsage });
        };

        return (
          <div>
            <h2>Операції зі списком</h2>
            <button onClick={() => runOperations(500)}>Запустити на 500 елементах</button>
            <button onClick={() => runOperations(1000)}>Запустити на 1000 елементах</button>
            <button onClick={() => runOperations(10000)}>Запустити на 10 000 елементах</button>

            <h3>Результати:</h3>
            {results.times && (
              <ul>
                {results.times.map((result, index) => (
                  <li key={index}>
                    Операція: {result.operation}, Час виконання: {result.time} мс
                  </li>
                ))}
              </ul>
            )}
            {results.memoryUsage && <p>Використання пам'яті: {results.memoryUsage} MB</p>}
          </div>
        );
      }

export default ListComponent;
```

1.4 Текст файлу mapComponent.js React

```
import React, { useState } from 'react';
import { generateData } from '../../utils/dataGenerator'; // Импортируем генератор данных

class ListNode {
  constructor(value, next = null) {
    this.value = value;
    this.next = next;
  }
}

class LinkedList {
  constructor() {
    this.head = null;
  }

  add(value) {
    const newNode = new ListNode(value);
    if (!this.head) {
      this.head = newNode;
    } else {
      let current = this.head;
      while (current.next) {
        current = current.next;
      }
      current.next = newNode;
    }
  }

  searchFirstAndLastThird() {
    const arr = this.getArrayFromList();
    const oneThird = Math.floor(arr.length / 3);
    const firstThird = arr.slice(0, oneThird);
    const lastThird = arr.slice(-oneThird);
    return { firstThird, lastThird };
  }
```

```
deleteEverySecondElement() {
  let current = this.head;
  let index = 0;

  while (current && current.next) {
    if (index % 2 === 0) {
      current.next = current.next.next;
    }
    current = current.next;
    index++;
  }
}

bubbleSort() {
  let arr = this.getArrayFromList();
  let n = arr.length;
  let swapped;

  do {
    swapped = false;
    for (let i = 0; i < n - 1; i++) {
      if (arr[i].id > arr[i + 1].id) {
        [arr[i], arr[i + 1]] = [arr[i + 1], arr[i]];
        swapped = true;
      }
    }
    n--;
  } while (swapped);

  this.fromArrayToList(arr);
}

mergeSort() {
  let arr = this.getArrayFromList();

  const merge = (left, right) => {
    let result = [];
    let leftIndex = 0;
    let rightIndex = 0;

    while (leftIndex < left.length && rightIndex < right.length) {
      if (left[leftIndex].id < right[rightIndex].id) {
        result.push(left[leftIndex]);
        leftIndex++;
      } else {
        result.push(right[rightIndex]);
        rightIndex++;
      }
    }
    return result.concat(left.slice(leftIndex)).concat(right.slice(rightIndex));
  };

  const mergeSortRec = (arr) => {
    if (arr.length <= 1) return arr;

    const mid = Math.floor(arr.length / 2);
    const left = mergeSortRec(arr.slice(0, mid));
    const right = mergeSortRec(arr.slice(mid));

    return merge(left, right);
  };

  arr = mergeSortRec(arr);
  this.fromArrayToList(arr);
}

getArrayFromList() {
  const arr = [];
  let current = this.head;
  while (current) {
    arr.push(current.value);
    current = current.next;
  }
  return arr;
}
```

```
 fromArrayToList(arr) {
  this.head = null;
  for (let i = 0; i < arr.length; i++) {
   this.add(arr[i]);
  }
 }
}

function ListComponent() {
 const [results, setResults] = useState([]);

 const measureExecutionTime = (func, ...args) => {
  const start = performance.now();
  func(...args);
  const end = performance.now();
  return end - start;
 };

 const measureMemoryUsage = () => {
  if (performance.memory) {
   return performance.memory.usedJSHeapSize / (1024 * 1024); // в MB
  } else {
   return 'Memory API не підтримується';
  }
 };

 const runOperations = (size) => {
  const list = new LinkedList();
  const data = generateData(size); // Генерация данных
  const times = [];
  const memoryUsage = [];

  // Додавання елементів
  const addTime = measureExecutionTime(() => {
   data.forEach((item) => list.add(item));
  });
  times.push({ operation: 'Додавання', time: addTime });

  // Пошук першої і останньої третини
  const searchTime = measureExecutionTime(() => {
   const { firstThird, lastThird } = list.searchFirstAndLastThird();
   console.log('Перша третина:', firstThird);
   console.log('Остання третина:', lastThird);
  });
  times.push({ operation: 'Пошук', time: searchTime });

  // Видалення кожного другого елемента
  const deleteTime = measureExecutionTime(() => {
   list.deleteEverySecondElement();
  });
  times.push({ operation: 'Видалення кожного другого елемента', time: deleteTime });

  // Bubble Sort
  const bubbleSortTime = measureExecutionTime(() => {
   list.bubbleSort();
  });
  times.push({ operation: 'Bubble Sort', time: bubbleSortTime });

  // Merge Sort
  const mergeSortTime = measureExecutionTime(() => {
   list.mergeSort();
  });
  times.push({ operation: 'Merge Sort', time: mergeSortTime });

  memoryUsage.push(measureMemoryUsage());

  setResults({ times, memoryUsage });
 };

 return (
  <div>
   <h2>Операції зі списком</h2>
   <button onClick={() => runOperations(500)}>Запустити на 500 елементах</button>
   <button onClick={() => runOperations(1000)}>Запустити на 1000 елементах</button>
```

```
    <button onClick={() => runOperations(10000)}>Запустити на 10 000 елементах</button>

    <h3>Результати:</h3>
    {results.times && (
     <ul>
      {results.times.map((result, index) => (
       <li key={index}>
        Операція: {result.operation}, Час виконання: {result.time} мс
       </li>
      ))}
     </ul>
    )}
    {results.memoryUsage && <p>Використання пам'яті: {results.memoryUsage} MB</p>}
   </div>
  );
}

export default ListComponent;
```

1.5 Текст файлу dataGenerator.js на React

```
import { faker } from '@faker-js/faker';

// Функция для генерации данных с использованием Faker.js
export const generateData = (size) => {
 const data = [];
 for (let i = 0; i < size; i++) {
  data.push({
   id: faker.string.uuid(), // Используем правильный метод uuid
   name: faker.person.fullName(), // Обновленный метод для имени
   address: faker.location.streetAddress(), // Обновленный метод для адреса
  });
 }
 return data;
};
```

1.6 Текст файлу App.js на React-native

```
import React, { useState } from 'react';
import { View, Text, Button, StyleSheet } from 'react-native';
import ArrayComponent from './components/ArrayComponent';
import ListComponent from './components/ListComponent';
import MapComponent from './components/MapComponent';

export default function App() {
 const [selectedComponent, setSelectedComponent] = useState(null);

 const renderSelectedComponent = () => {
  switch (selectedComponent) {
   case 'array':
    return <ArrayComponent />;
   case 'list':
    return <ListComponent />;
   case 'map':
    return <MapComponent />;
   default:
    return <Text style={styles.instructions}>Оберіть компонент для тестування</Text>;
  }
 };

 return (
  <View style={styles.container}>
   <Text style={styles.header}>Тестування продуктивності</Text>
   <View style={styles.buttonContainer}>
    <Button title="Тестувати масив" onPress={() => setSelectedComponent('array')} />
    <Button title="Тестувати список" onPress={() => setSelectedComponent('list')} />
    <Button title="Тестувати мапу" onPress={() => setSelectedComponent('map')} />
   </View>
   {renderSelectedComponent()}
  </View>
```

```
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 20,
    backgroundColor: '#fff',
  },
  header: {
    fontSize: 24,
    fontWeight: 'bold',
    marginBottom: 20,
    textAlign: 'center',
  },
  buttonContainer: {
    flexDirection: 'row',
    justifyContent: 'space-around',
    marginBottom: 20,
  },
  instructions: {
    fontSize: 18,
    textAlign: 'center',
    marginTop: 20,
  },
});
```

1.7 Текст файлу ArrayComponent.js на React-native

```
import React, { useState } from 'react';
import { View, Text, TouchableOpacity, FlatList, StyleSheet } from 'react-native';
import { generateData } from '../utils/generateData'; // Імпортуємо функцію генерації даних

function ArrayComponent() {
  const [results, setResults] = useState({ times: [], memoryUsage: null });

  const measureExecutionTime = (func, ...args) => {
    const start = performance.now();
    func(...args);
    const end = performance.now();
    return end - start;
  };

  const measureMemoryUsage = () => {
    if (performance.memory) {
      return performance.memory.usedJSHeapSize / (1024 * 1024); // у MB
    } else {
      return 'Memory API не підтримується';
    }
  };

  const deleteEverySecondElement = (arr) => {
    return arr.filter((_, index) => index % 2 === 0);
  };

  const searchFirstAndLastThird = (arr) => {
    const oneThird = Math.floor(arr.length / 3);
    const firstThird = arr.slice(0, oneThird);
    const lastThird = arr.slice(-oneThird);
    return { firstThird, lastThird };
  };

  const bubbleSort = (arr) => {
    let n = arr.length;
    let swapped;
    do {
      swapped = false;
      for (let i = 0; i < n - 1; i++) {
        if (arr[i].id > arr[i + 1].id) {
          [arr[i], arr[i + 1]] = [arr[i + 1], arr[i]];
          swapped = true;
        }
```

```
    }
    n--;
  } while (swapped);
  return arr;
};

const mergeSort = (arr) => {
  if (arr.length <= 1) return arr;

  const mid = Math.floor(arr.length / 2);
  const left = mergeSort(arr.slice(0, mid));
  const right = mergeSort(arr.slice(mid));

  return merge(left, right);
};

const merge = (left, right) => {
  let result = [];
  let leftIndex = 0;
  let rightIndex = 0;

  while (leftIndex < left.length && rightIndex < right.length) {
    if (left[leftIndex].id < right[rightIndex].id) {
      result.push(left[leftIndex]);
      leftIndex++;
    } else {
      result.push(right[rightIndex]);
      rightIndex++;
    }
  }

  return result.concat(left.slice(leftIndex)).concat(right.slice(rightIndex));
};

const runOperations = (size) => {
  let array = generateData(size);
  const times = [];
  const memoryUsage = [];

  const addTime = measureExecutionTime(() => {
    array = generateData(size);
  });
  times.push({ operation: 'Додавання', time: addTime });

  const searchTime = measureExecutionTime(() => {
    const { firstThird, lastThird } = searchFirstAndLastThird(array);
  });
  times.push({ operation: 'Пошук', time: searchTime });

  const deleteTime = measureExecutionTime(() => {
    array = deleteEverySecondElement(array);
  });
  times.push({ operation: 'Видалення кожного другого елемента', time: deleteTime });

  const bubbleSortTime = measureExecutionTime(() => {
    bubbleSort([...array]);
  });
  times.push({ operation: 'Bubble Sort', time: bubbleSortTime });

  const mergeSortTime = measureExecutionTime(() => {
    mergeSort([...array]);
  });
  times.push({ operation: 'Merge Sort', time: mergeSortTime });

  memoryUsage.push(measureMemoryUsage());

  setResults({ times, memoryUsage });
};

return (
  <View style={styles.container}>
    <Text style={styles.header}>Операції з масивом</Text>
    <View style={styles.buttonsContainer}>
      <TouchableOpacity style={styles.button} onPress={() => runOperations(500)}>
        <Text style={styles.buttonText}>Запустити на 500 елементах</Text>
```

```
      </TouchableOpacity>
      <TouchableOpacity style={styles.button} onPress={() => runOperations(1000)}>
        <Text style={styles.buttonText}>Запустити на 1000 елементах</Text>
      </TouchableOpacity>
      <TouchableOpacity style={styles.button} onPress={() => runOperations(10000)}>
        <Text style={styles.buttonText}>Запустити на 10 000 елементах</Text>
      </TouchableOpacity>
    </View>

    <FlatList
      data={results.times}
      keyExtractor={(item, index) => index.toString()}
      renderItem={({ item }) => (
        <Text style={styles.resultItem}>
          Операція: {item.operation}, Час виконання: {item.time} мс
        </Text>
      )}
    />
    <Text style={styles.resultItem}>Використання пам'яті: {results.memoryUsage} MB</Text>
  </View>
 );
}

const styles = StyleSheet.create({
 container: {
  padding: 10,
 },
 header: {
  fontSize: 20,
  fontWeight: 'bold',
  marginBottom: 10,
 },
 buttonsContainer: {
  flexDirection: 'row',
  justifyContent: 'space-around',
  marginBottom: 20,
 },
 button: {
  backgroundColor: '#4CAF50',
  padding: 10,
  borderRadius: 5,
 },
 buttonText: {
  color: 'white',
 },
 resultItem: {
  marginVertical: 5,
 },
});

export default ArrayComponent;
```

1.8 Текст файлу ListComponent.js на React-native

```
import React, { useState } from 'react';
import { View, Text, TouchableOpacity, FlatList, StyleSheet } from 'react-native';
import { generateData } from '../utils/generateData';

function ListComponent() {
 const [results, setResults] = useState({ times: [], memoryUsage: null });

 const measureExecutionTime = (func, ...args) => {
  const start = performance.now();
  func(...args);
  const end = performance.now();
  return end - start;
 };

 const measureMemoryUsage = () => {
  if (performance.memory) {
   return performance.memory.usedJSHeapSize / (1024 * 1024); // у MB
  } else {
   return 'Memory API не підтримується';
```

```
  }
};

class ListNode {
  constructor(value, next = null) {
    this.value = value;
    this.next = next;
  }
}

class LinkedList {
  constructor() {
    this.head = null;
  }

  add(value) {
    const newNode = new ListNode(value);
    if (!this.head) {
      this.head = newNode;
    } else {
      let current = this.head;
      while (current.next) {
        current = current.next;
      }
      current.next = newNode;
    }
  }

  search(id) {
    let current = this.head;
    while (current) {
      if (current.value.id === id) return true;
      current = current.next;
    }
    return false;
  }

  deleteEverySecond() {
    let current = this.head;
    while (current && current.next) {
      current.next = current.next.next;
      current = current.next;
    }
  }

  bubbleSort() {
    let swapped;
    do {
      swapped = false;
      let current = this.head;
      while (current && current.next) {
        if (current.value.id > current.next.value.id) {
          [current.value, current.next.value] = [current.next.value, current.value];
          swapped = true;
        }
        current = current.next;
      }
    } while (swapped);
  }

  mergeSort(head = this.head) {
    if (!head || !head.next) return head;

    const getMiddle = (head) => {
      let slow = head;
      let fast = head.next;
      while (fast && fast.next) {
        slow = slow.next;
        fast = fast.next.next;
      }
      return slow;
    };

    const merge = (left, right) => {
      const dummy = new ListNode(null);
```

```
    let tail = dummy;
    while (left && right) {
      if (left.value.id < right.value.id) {
        tail.next = left;
        left = left.next;
      } else {
        tail.next = right;
        right = right.next;
      }
      tail = tail.next;
    }
    tail.next = left ? left : right;
    return dummy.next;
  };

  const middle = getMiddle(head);
  const nextToMiddle = middle.next;
  middle.next = null;

  const left = this.mergeSort(head);
  const right = this.mergeSort(nextToMiddle);

  return merge(left, right);
 }
}

const runOperations = (size) => {
 const list = new LinkedList();
 const times = [];

 const data = generateData(size);

 const addTime = measureExecutionTime(() => {
  data.forEach((item) => list.add(item));
 });
 times.push({ operation: 'Додавання', time: addTime });

 const searchTime = measureExecutionTime(() => {
  list.search(data[size - 1].id);
 });
 times.push({ operation: 'Пошук', time: searchTime });

 const deleteTime = measureExecutionTime(() => {
  list.deleteEverySecond();
 });
 times.push({ operation: 'Видалення кожного другого елемента', time: deleteTime });

 const bubbleSortTime = measureExecutionTime(() => {
  list.bubbleSort();
 });
 times.push({ operation: 'Bubble Sort', time: bubbleSortTime });

 const mergeSortTime = measureExecutionTime(() => {
  list.head = list.mergeSort();
 });
 times.push({ operation: 'Merge Sort', time: mergeSortTime });

 // Вимірюємо використання пам'яті
 const memoryUsage = measureMemoryUsage();

 setResults({ times, memoryUsage });
};

return (
 <View style={styles.container}>
  <Text style={styles.header}>Операції зі списком</Text>
  <View style={styles.buttonsContainer}>
   <TouchableOpacity style={styles.button} onPress={() => runOperations(500)}>
    <Text style={styles.buttonText}>Запустити на 500 елементах</Text>
   </TouchableOpacity>
   <TouchableOpacity style={styles.button} onPress={() => runOperations(1000)}>
    <Text style={styles.buttonText}>Запустити на 1000 елементах</Text>
   </TouchableOpacity>
   <TouchableOpacity style={styles.button} onPress={() => runOperations(10000)}>
    <Text style={styles.buttonText}>Запустити на 10 000 елементах</Text>
```

```
      </TouchableOpacity>
    </View>

    <FlatList
      data={results.times}
      keyExtractor={(item, index) => index.toString()}
      renderItem={({ item }) => (
        <Text style={styles.resultItem}>
          Операція: {item.operation}, Час виконання: {item.time} мс
        </Text>
      )}
    />
    <Text style={styles.resultItem}>Використання пам'яті: {results.memoryUsage} MB</Text>
  </View>
 );
}

const styles = StyleSheet.create({
 container: {
  padding: 10,
 },
 header: {
  fontSize: 20,
  fontWeight: 'bold',
  marginBottom: 10,
 },
 buttonsContainer: {
  flexDirection: 'row',
  justifyContent: 'space-around',
  marginBottom: 20,
 },
 button: {
  backgroundColor: '#4CAF50',
  padding: 10,
  borderRadius: 5,
 },
 buttonText: {
  color: 'white',
 },
 resultItem: {
  marginVertical: 5,
 },
});

export default ListComponent;
```

1.9 Текст файлу MapComponent.js на React-native

```
import React, { useState } from 'react';
import { View, Text, TouchableOpacity, FlatList, StyleSheet } from 'react-native';
import { generateData } from '../utils/generateData'; // Імпорт функції генерації даних

function MapComponent() {
 const [results, setResults] = useState({ times: [], memoryUsage: null });

 const measureExecutionTime = (func, ...args) => {
  const start = performance.now();
  func(...args);
  const end = performance.now();
  return end - start;
 };

 const measureMemoryUsage = () => {
  if (performance.memory) {
   return (performance.memory.usedJSHeapSize / (1024 * 1024)).toFixed(2); // в MB
  } else {
   return 'Memory API не підтримується';
  }
 };

 const deleteEverySecondElement = (map) => {
```

```
  let count = 0;
  for (let [key] of map) {
    if (count % 2 === 0) {
      map.delete(key);
    }
    count++;
  }
};

const searchFirstAndLastThird = (map) => {
  const keysArray = Array.from(map.keys());
  const oneThird = Math.floor(keysArray.length / 3);

  const firstThird = keysArray.slice(0, oneThird); // Перша третина
  const lastThird = keysArray.slice(-oneThird); // Остання третина

  console.log('Перша третина ключів:', firstThird);
  console.log('Остання третина ключів:', lastThird);
};

const bubbleSort = (map) => {
  const keysArray = Array.from(map.keys());
  let n = keysArray.length;
  let swapped;
  do {
    swapped = false;
    for (let i = 0; i < n - 1; i++) {
      if (keysArray[i] > keysArray[i + 1]) {
        [keysArray[i], keysArray[i + 1]] = [keysArray[i + 1], keysArray[i]]; // Міняємо місцями
        swapped = true;
      }
    }
    n--;
  } while (swapped);
  return new Map(keysArray.map((key) => [key, map.get(key)]));
};

const mergeSort = (map) => {
  const keysArray = Array.from(map.keys());

  const merge = (left, right) => {
    let result = [];
    let leftIndex = 0;
    let rightIndex = 0;

    while (leftIndex < left.length && rightIndex < right.length) {
      if (left[leftIndex] < right[rightIndex]) {
        result.push(left[leftIndex]);
        leftIndex++;
      } else {
        result.push(right[rightIndex]);
        rightIndex++;
      }
    }
    return result.concat(left.slice(leftIndex)).concat(right.slice(rightIndex));
  };

  const mergeSortRec = (arr) => {
    if (arr.length <= 1) return arr;
    const mid = Math.floor(arr.length / 2);
    const left = mergeSortRec(arr.slice(0, mid));
    const right = mergeSortRec(arr.slice(mid));
    return merge(left, right);
  };

  const sortedKeys = mergeSortRec(keysArray);
  return new Map(sortedKeys.map((key) => [key, map.get(key)]));
};

const runOperations = (size) => {
  const map = new Map();
  const times = [];

  // Генерація даних і додавання їх у map
  const data = generateData(size);
```

```
    const addTime = measureExecutionTime(() => {
      data.forEach((obj) => {
        map.set(obj.id, obj);
      });
    });
    times.push({ operation: 'Додавання', time: addTime });

    // Пошук першої і останньої третини
    const searchTime = measureExecutionTime(() => {
      searchFirstAndLastThird(map);
    });
    times.push({ operation: 'Пошук першої і останньої третини', time: searchTime });

    // Видалення кожного другого елемента
    const deleteTime = measureExecutionTime(() => {
      deleteEverySecondElement(map);
    });
    times.push({ operation: 'Видалення кожного другого елемента', time: deleteTime });

    // Bubble Sort
    const bubbleSortTime = measureExecutionTime(() => {
      const sortedMap = bubbleSort(map);
      console.log('Мапа після Bubble Sort:', sortedMap);
    });
    times.push({ operation: 'Bubble Sort', time: bubbleSortTime });

    // Merge Sort
    const mergeSortTime = measureExecutionTime(() => {
      const sortedMap = mergeSort(map);
      console.log('Мапа після Merge Sort:', sortedMap);
    });
    times.push({ operation: 'Merge Sort', time: mergeSortTime });

    const memoryUsage = measureMemoryUsage();
    setResults({ times, memoryUsage });
  };

  return (
    <View style={styles.container}>
      <Text style={styles.header}>Операції з мапою</Text>
      <View style={styles.buttonsContainer}>
        <TouchableOpacity style={styles.button} onPress={() => runOperations(500)}>
          <Text style={styles.buttonText}>Запустити на 500 елементах</Text>
        </TouchableOpacity>
        <TouchableOpacity style={styles.button} onPress={() => runOperations(1000)}>
          <Text style={styles.buttonText}>Запустити на 1000 елементах</Text>
        </TouchableOpacity>
        <TouchableOpacity style={styles.button} onPress={() => runOperations(10000)}>
          <Text style={styles.buttonText}>Запустити на 10 000 елементах</Text>
        </TouchableOpacity>
      </View>

      <Text style={styles.resultsHeader}>Результати:</Text>
      <FlatList
        data={results.times}
        keyExtractor={(item, index) => index.toString()}
        renderItem={({ item }) => (
          <Text style={styles.resultItem}>
            Операція: {item.operation}, Час виконання: {item.time.toFixed(2)} мс
          </Text>
        )}
      />
      {results.memoryUsage && (
        <Text style={styles.memoryUsage}>Використання пам'яті: {results.memoryUsage} MB</Text>
      )}
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 20,
    backgroundColor: '#fff',
  },
```

```
  header: {
    fontSize: 20,
    fontWeight: 'bold',
    marginBottom: 20,
  },
  buttonsContainer: {
    flexDirection: 'row',
    justifyContent: 'space-around',
    marginBottom: 20,
  },
  button: {
    backgroundColor: '#4CAF50',
    padding: 10,
    borderRadius: 5,
    marginVertical: 5,
  },
  buttonText: {
    color: 'white',
    fontSize: 14,
    textAlign: 'center',
  },
  resultsHeader: {
    fontSize: 18,
    fontWeight: 'bold',
    marginTop: 20,
  },
  resultItem: {
    fontSize: 16,
    marginVertical: 5,
  },
  memoryUsage: {
    fontSize: 16,
    marginTop: 10,
  },
});

export default MapComponent;
```

1.10 Текст файлу dataGenerator.js на React-native

```
import { faker } from '@faker-js/faker';

// Функция для генерации массива объектов
export const generateData = (size) => {
  const data = [];
  for (let i = 0; i < size; i++) {
    const obj = {
      id: faker.string.uuid(),
      name: faker.person.fullName(),
      address: faker.location.streetAddress(),
    };
    data.push(obj);
  }
  return data;
};
```

ДОДАТОК В

Керівництво користувача

ЗАТВЕРДЖУЮ
Проректор Українського державного
університету науки і технологій
Анатолій РАДКЕВИЧ

«АНАЛІЗ РЕСУРСОЄМНОСТІ ВЕБ-ДОДАТКІВ РОЗРОБЛЕНИХ ЗА ДОПОМОГОЮ REACT ТА REACT-NATIVE»

Керівництво користувача
44165850.01434 – 01 І3 01

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН
Керівник розробки
_____Олександр ІВАНОВ
Виконавець
_____Кирило ДОРОГОКУПЛЯ
Нормоконтролер
_____Світлана ВОЛКОВА

2025

ЗАТВЕРДЖЕНО
44165850.01434-01 ІЗ 01

«АНАЛІЗ РЕСУРСОЄМНОСТІ ВЕБ-ДОДАТКІВ РОЗРОБЛЕНИХ ЗА ДОПОМОГОЮ REACT ТА REACT-NATIVE»

Керівництво користувача
44165850.01434 – 01 ІЗ 01

Листів 9

2025

44165850.01434-01 І3 01

АНОТАЦІЯ

Документ 1116130.01434 – 01 І3 01 «Аналіз ресурсоємності веб-додатків, розроблених за допомогою React та React Native. Керівництво користувача».

Програми написані на мові JavaScript, на фреймворках React та React-native, з використанням бібліотек Faker.js, PerformanceAPI програмному середовищі Visual Studio Code.

ЗМІСТ

1 ВВЕДЕННЯ

Програмний засіб «Аналіз ресурсоємності веб-додатків, розроблених за допомогою React та React Native» призначений для вивчення та аналізу продуктивності веб-додатків. Головною метою розробки є створення інструментарію, що дозволяє досліджувати час виконання операцій та використання пам’яті при роботі з різними обсягами даних і виконанні типових операцій: додавання, видалення, пошуку та сортування.

Цей продукт орієнтований на науковців, розробників та фахівців у галузі програмного забезпечення, які прагнуть дослідити ефективність використання технологій React і React Native для створення веб-додатків. Він також підходить для практичного вибору найкращої технології залежно від специфічних вимог проекту.

Програмний засіб простий у використанні, не потребує значного досвіду в розробці програмного забезпечення та може бути використаний навіть недосвідченими користувачами. Використання програми не вимагає додаткової експлуатаційної документації, що робить її доступною для широкого кола зацікавлених осіб.

2 ПРИЗНАЧЕННЯ ТА УМОВИ ЗАСТОСУВАННЯ

Функціональним призначенням додатку є проведення аналізу ресурсоємності веб-додатків, розроблених за допомогою React та React Native. Додаток дозволяє оцінити продуктивність і використання пам'яті при виконанні типових операцій, таких як додавання, видалення, пошук і сортування даних, у різних умовах та на різних обсягах даних. Це сприяє вибору оптимальної технології для розробки веб-додатків залежно від конкретних завдань.

Експлуатаційне призначення додатку «Аналіз ресурсоємності веб-додатків, розроблених за допомогою React та React Native» полягає в забезпеченні інструментів для тестування і порівняння ефективності цих фреймворків. Це дозволяє розробникам і науковцям приймати обґрунтовані рішення щодо вибору технології для своїх проектів.

Для забезпечення сталого функціонування програми необхідно дотримуватися таких умов:

- наявність стабільного інтернет-з’єднання для доступу до зовнішніх бібліотек і сервісів;
- встановлення всіх залежностей через відповідний менеджер пакетів (npm або yarn);
- наявність сумісного середовища розробки для запуску веб- та мобільних додатків.

Додаток, що розробляється, розрахований на використання на пристроях що мають:

- процесор з тактовою частотою не нижче 2.5 ГГц;
- не менше 4ГБ оперативної пам’яті.

3 ПІДГОТОВКА ДО РОБОТИ

Попередня установка

– Інсталяція середовища розробки: Завантажте та встановіть Node.js із офіційного сайту. Разом із ним автоматично буде встановлено npm. Для тестування мобільних додатків налаштуйте Android Studio або Xcode залежно від операційної системи.

– Склонування проекту:
Отримайте вихідний код із репозиторію (GitHub або інше сховище). Виконайте команду:

git clone https://github.com/KirilDora/[react-performance або react-native-performance]

cd [react-performance або react-native-performance]

– Установка залежностей:
Перейдіть до папки проекту та виконайте команду:

npm install або *yarn install*

Запуск програми:

– запуск React-додатку виконайте команду:

npm start

Додаток буде доступний у браузері за адресою http://localhost:3000.

– запуск React Native-додатку виконайте команду:

npm run start -w

4 ОПИС ОПЕРАЦІЙ

Після встановлення та запуску програмного продукту "Аналіз ресурсоємності веб-додатків, розроблених за допомогою React та React Native", користувач отримує доступ до наступних елементів інтерфейсу (рис.1 та рис.2):

– вибір структури даних: користувач може вибрати тип структури даних (масив, список, мапа), яку буде використовувати додаток для аналізу;

– поле введення кількості елементів: забезпечує можливість вказати обсяг даних для обробки (500, 1000, 10000 елементів);

– секція результатів: відображає час виконання кожної операції (додавання, видалення, пошук, сортування) для обраної структури даних.

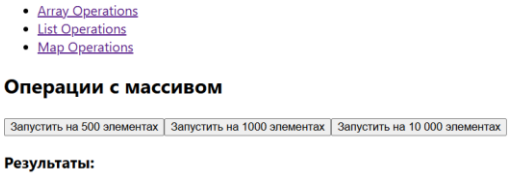

Рисунок 1 – інтерфейс додатку на React

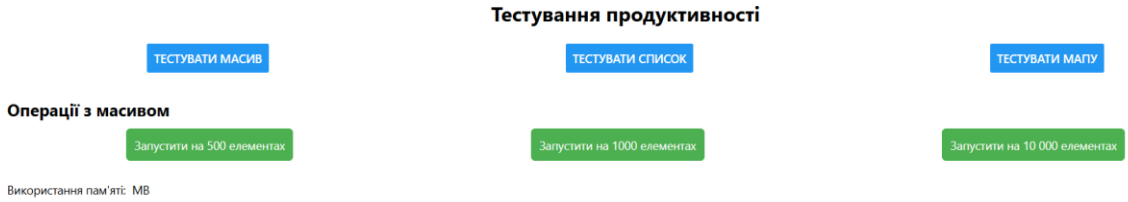

Рисунок 2 – інтерфейс додатку на React-native

5 АВАРІЙНІ СИТУАЦІЇ

Якщо програмний засіб буде запускатися з пристрою де немає інтернет з’єднання, додаток виведе на екран повідомлення про помилку та завершить роботу.

Якщо програмний засіб під час роботи буде поводити некоректно чи із помилкою, необхідно перезапустити додаток для продовження роботи.

6 РЕКОМЕНДАЦІЇ ЩОДО ЗАСТОСУВАННЯ

Після встановлення додатку на пристрій його необхідно запустити за допомогою командного рядка.

Після запуску додатку перед користувачем відкриється веб-сторінка із головним екраном з усім необхідним функціоналом.

Після запуску додатку перед користувачем відкриється головний екран, де буде запропоновано вибрати:

- структуру даних для аналізу (масив, список, мапа);
- кількість елементів для обробки (500, 1000, 10 000).

Після ознайомлення із функціями додатку, його роботу можна звершити натиснувши апаратну кнопку закриття вкладки та натиснув в командному рядку комбінацію *CTRL + C*.