КОМП'ЮТЕРНІ НАУКИ

УДК 004.4
DOI https://doi.org/10.32782/2521-6643-2024-1-67.5

Безверхий О. І., доктор фізико-математичних наук, професор, професор кафедри інформаційних систем і технологій Національного транспортного університету
ORCID: 0000-0002-0834-6335

Куценко О. І., аспірант кафедри інформаційних систем і технологій Національного транспортного університету
ORCID: 0000-0003-0047-4874

ШЛЯХИ ОПТИМІЗАЦІЇ КРОСПЛАТФОРМЕННИХ ДОДАТКІВ ІЗ ВИКОРИСТАННЯМ БІБЛІОТЕКИ REACT ТА ФРЕЙМВОРКУ REACT NATIVE

У сучасному світі, зі зростанням використання мобільних пристроїв, важливість створення кросплатформених додатків, що забезпечують високу продуктивність та якість користувацького досвіду, значно зросла. Удосконалення створення таких додатків з використанням React та React Native відкриває широкі можливості для розробників завдяки можливості перевикористання коду, високій швидкості розробки та легкості впровадження інновацій і набуває особливої актуальності, оскільки вона стосується не тільки технологічних аспектів розробки, але й економічної ефективності, якості користувацького досвіду та швидкості впровадження інновацій на ринок програмного забезпечення. Відкритий код та активне спілкування у спільноті дозволяють швидко виявляти та усувати проблеми, розробляти нові функції та покращення, а також сприяють обміну знаннями та досвідом між розробниками. Це створює позитивне середовище для інновацій та росту, сприяє швидкому прогресу технологій та підвищує якість кінцевих продуктів.

В роботі проведено аналіз ключових аспектів та викликів, пов'язаних із створенням кросплатформених додатків, аналізуються основні види оптимізацій на різних рівнях розробки, включаючи компонентний рівень, рівень стану та даних, рівень роботи з API та зовнішніми даними, а також рівень загрузки та коду. Особлива увага приділяється інтеграції синхронного та асинхронного рендерингу для досягнення оптимальної продуктивності та користувацького досвіду. Висвітлюються переваги гнучкого управління станом за допомогою стейт-менеджерів, а також важливість оптимізації зображень та медіа для підвищення загальної продуктивності веб-сайтів.

В статті відзначена роль спільноти розробників у процесі удосконалення та інновацій. Розробники, які володіють цими технологіями та оптимізаціями їх застосування, можуть створювати продукти, що не просто відповідають вимогам часу, а формують нові стандарти у сфері мобільної та веб-розробки. Подальше дослідження та інтеграція новітніх технологій у процес розробки кросплатформенних додатків буде мати значний вплив на індустрію програмного забезпечення.

Ключові слова: *React, React Native, кросплатформені додатки, оптимізація, розробка програмного забезпечення.*

Bezverhiy O. I., Kutsenko O. I. Ways of optimizing cross-platform application using the React library and the React Native framework

In today's world, with the growing use of mobile devices, the importance of creating cross-platform applications that provide high performance and quality of user experience has increased significantly. Improving the creation of such applications using React and React Native opens up wide opportunities for developers due to the possibility of code reuse, high development speed and ease of innovation, and is especially relevant because it concerns not only the technological aspects of development, but also economic efficiency, quality of user experience and the speed of introducing innovations to the software market. Open source and active community communication allow rapid identification and resolution of issues, development of new features and enhancements, and the sharing of knowledge and experience between developers. This creates a positive environment for innovation and growth, promotes rapid technological progress, and improves the quality of end products.

The paper analyzes the key aspects and challenges associated with the creation of cross-platform applications, analyzes the main types of optimizations at different levels of development, including the component level, the state and data level, the level of working with APIs and external data, as well as the loading and code level. Special attention is paid to the integration of synchronous and asynchronous rendering to achieve optimal performance and user experience. Highlights the benefits of flexible state management using state managers, as well as the importance of image and media optimization to improve overall website performance.

The article notes the role of the developer community in the process of improvement and innovation. Developers who master these technologies and optimize their application can create products that not only meet the requirements of the time, but also form new standards in the field of mobile and web development. Further research and integration of the latest technologies into the development process of cross-platform applications will have a significant impact on the software industry.
Key words: *React, React Native, cross-platform applications, optimization, software development.*

Постановка проблеми. Використання бібліотеки React та React-native для створення крос платформених додатків випливає з ряду тенденцій:

1. Зростання мобільного трафіку та використання мобільних додатків. У сучасному світі спостерігається стійке зростання кількості користувачів мобільних пристроїв, що спонукає до розробки додатків, здатних задовольнити різноманітні потреби цієї аудиторії. Кросплатформність стає ключовою відповіддю на цей виклик, дозволяючи розробникам створювати додатки, що ефективно функціонують на різних платформах.

2. Необхідність скорочення часу та витрат на розробку. Розробка окремих версій додатків для кожної платформи є часомісткою та дорогою. Використання React та React Native дозволяє значно оптимізувати ресурси, оскільки розробники можуть використовувати одну та ту ж кодову базу для створення додатків, що працюють на різних операційних системах.

3. Попит на високу продуктивність та якість додатків.

4. Необхідність постійного оновлення та підтримки додатків. Умови ринку та вимоги користувачів постійно змінюються, тому додатки потребують регулярного оновлення та модернізації.

Наукові дослідження в галузі оптимізації застосування JavaScript, розробка нових патернів розробки, покращення архітектури додатків та інтеграція з іншими технологіями і платформами є критично важливими для досягнення цієї мети. Спільними зусиллями науковців, розробників і спільноти можна досягти значного прогресу у створенні ефективних, доступних та високоякісних кросплатформних додатків, що задовольнятимуть потреби сучасних користувачів і ринку програмного забезпечення.

Аналіз основних видів оптимізацій React-додатку. Оптимізація React-додатків є ключовим елементом для підвищення продуктивності, забезпечення швидкості роботи інтерфейсу та покращення загального користувацького досвіду. Існує кілька рівнів, на яких можливо виконати оптимізацію React-додатків, кожен з яких вимагає застосування специфічних підходів і технік. Розглянемо основні види оптимізацій на різних рівнях:

1. Компонентний рівень

Перевикористання компонентів

– Створення перевикористовувальних компонентів для зменшення дублювання коду та підвищення читабельності та підтримки коду.

Попередження непотрібних рендерів

– Використання React.memo для класових компонентів та React.PureComponent для функціональних компонентів дозволяє уникнути непотрібних рендерів, порівнюючи props та state.

Якщо компонент відображає той самий результат з тими самими пропсами та станом, можна обгорнути його у виклик React.memo для підвищення продуктивності в деяких випадках шляхом запам'ятовування результату. Це означає, що React пропустить рендеринг компоненту та повторно використає останній результат рендерингу.

React.memo тільки перевіряє чи змінилися пропси. Якщо функція, згорнута у React.memo, має useState або useContext хуки в своїй імплементації, вона все ще буде перендеритися при зміні стана або контекста.[1]

За замовчуванням він тільки поверхово порівнює складні об'єкти, що знаходяться в об'єкті пропсів. Якщо ви хочете контролювати процес порівняння, ви також можете надати користувальницьку функцію для порівняння помістивши її другим аргументом.

Використання shouldComponentUpdate

– Для класових компонентів, метод shouldComponentUpdate дозволяє контролювати процес рендерінга шляхом порівняння поточних і наступних props та state.

Асинхронний рендеринг з React.Suspense та лінива загрузка:

– React.Suspense та React.lazy пропонують механізм для асинхронного рендеринга компонентів, що дозволяє компонентам чекати завантаження необхідних даних або інших компонентів перед їх відображенням. Це особливо корисно для покращення продуктивності шляхом лінивої загрузки компонентів, які не потрібні користувачеві відразу після завантаження додатку.

React.lazy приймає функцію, яка має викликати динамічний import(). Вона має повернути Promise, який при вирішенні поверне модуль з default-експортом, який містить React-компонент.

Ледачий компонент потім повинен відрендеритися у тілі компонента Suspense. Це дозволяє нам показати резервний контент (наприклад, індикатор завантаження), поки ми чекаємо на завантаження ледачого компонента.[2]

Використання React.memo та React.PureComponent для запобігання непотрібних рендерів та методу shouldComponentUpdate для контролю оновлень компонентів.

2. Рівень Стану та Даних

Управління Станом

– Ефективне управління станом, використовуючи контексти або стейт-менеджери типу Redux чи MobX, може значно зменшити кількість рендерів та спростити потік даних.

Бібліотека Redux використовується для керування станом додатку, що дозволяє зменшити кількість коду та зробити додаток більш простим для розуміння та розширення [3]

Розробка додатку на Redux без використання Redux toolkit може бути досить складною та вимагати багато часу. Необхідно вручну створювати дії, редуктори та створювати зв'язки між ними. З іншого боку, Redux toolkit має багато готових функцій, таких як createSlice, яка дозволяє автоматично створювати дії та редуктори для конкретного сегмента стану додатку. Це значно зменшує кількість написаного коду та спрощує розробку [4]

Лінива Загрузка Даних

– Застосування лінивої загрузки (lazy loading) для даних та компонентів може покращити час завантаження додатку та ефективність використання ресурсів.

Асинхронне Завантаження Даних:

– Використання хуків, наприклад useEffect, для асинхронного завантаження даних дозволяє ініціювати запити до API або баз даних без блокування рендеринга компонентів. Це дозволяє інтерфейсу залишатися відгуковим, поки дані завантажуються, та зменшує відчуття затримок у користувачів.

На відміну від componentDidMount і componentDidUpdate, функція передана в useEffect запускається після розмітки та рендеру, протягом відкладеної події. Це робить хук підходящим для багатьох поширених побічних ефектів, таких як налаштування підписок та обробників подій, оскільки більшість типів роботи не повинні блокувати оновлення екрану браузером. [5]

Використання Конкурентного Режиму (Concurrent Mode):

– Конкурентний режим у React надає більш гнучкий спосіб для асинхронного рендеринга інтерфейсів. Він дозволяє React розпочати рендеринг змін в інтерфейсі без того, щоб блокувати основний потік, тим самим підвищуючи відгуковість додатку та забезпечуючи більш гладке оновлення інтерфейсу.

Оптимізація на рівні стану та даних включає ефективне управління станом за допомогою контекстів чи стейт-менеджерів, що сприяє зменшенню рендерів та спрощенню потоку даних. Лінива загрузка даних та компонентів покращує час завантаження додатку. Асинхронне завантаження даних через хуки, як-от useEffect, дозволяє здійснювати запити без блокування рендеринга, підтримуючи інтерфейс відгуковим. Конкурентний режим підвищує відгуковість, дозволяючи асинхронний рендеринг змін.

3. Рівень Роботи з API та Зовнішніми Даними

Кешування Відповідей

– Кешування відповідей від сервера може зменшити кількість запитів до сервера та прискорити відображення даних користувачу.

Оптимізація Запитів

– Уникайте надмірних або непотрібних запитів до сервера, агрегуючи дані або використовуючи debounce/throttle для обробки подій.

На рівні роботи з API та зовнішніми даними, кешування відповідей від сервера може значно зменшити навантаження на сервер та прискорити процес завантаження даних для користувача. Оптимізація запитів через уникнення надмірних або непотрібних запитів, а також використання методів агрегації даних або debounce/throttle для обробки подій, дозволяє підвищити ефективність взаємодії з сервером та оптимізувати загальну продуктивність додатку

4. Рівень Загрузки та Коду

Code Splitting

– Використання розділення коду (code splitting) дозволяє розділити код на менші частини, які можуть бути завантажені за потреби, зменшуючи час завантаження додатку.

Лінива Загрузка Компонентів

– React.lazy та Suspense дозволяють організувати ліниву загрузку компонентів, що не є критичними для початкового рендеринга.

Використання Web Workers для Обробки Важких Завдань

– Для обробки складних обчислень без блокування основного потоку UI, можна використовувати Web Workers, що дозволяє покращити відгуковість інтерфейсу.

На рівні загрузки та коду, оптимізація включає в себе такі підходи як розділення коду (code splitting), що дозволяє зменшити час завантаження додатку шляхом поділу коду на менші частини, які завантажуються лише за потреби. Лінива загрузка компонентів за допомогою React.lazy та Suspense покращує продуктивність, оскільки важливі компоненти завантажуються тільки коли це необхідно. Використання Web Workers для обробки складних обчислень допомагає покращити відгуковість інтерфейсу, не блокуючи основний потік UI.

5. Оптимізація Зображень та Медіа

– Компресія зображень, використання форматів, оптимізованих для WEB (наприклад, WebP), та лінива загрузка медіа можуть значно покращити продуктивність за рахунок зменшення обсягу переданих даних.

Оптимізація зображень та медіа є важливою для підвищення продуктивності веб-сайтів. Компресія зображень, використання форматів, які оптимізовані для WEB, таких як WebP, та лінива загрузка медіа можуть значно зменшити обсяг переданих даних і таким чином покращити час завантаження сторінок та загальну продуктивність сайту.

Мета статті: аналіз потенціалу та особливостей використання шляхів оптимізації React і React Native для розробки кросплатформених додатків, а також аналіз основних напрямків оптимізації.

Оптимізація React-додатків на всіх цих рівнях дозволяє досягти значного підвищення продуктивності та забезпечити кращий досвід користувача. Важливо зауважити, що оптимізація є процесом, який потребує постійного аналізу, тестування та вдосконалення.

Застосування синхронного та асинхронного рендерингу для оптимізації додатків. Використання синхронного та асинхронного рендерингу у React повинно бути збалансоване, щоб забезпечити оптимальну продуктивність та користувацький досвід. Синхронний рендеринг може бути ефективним для відображення контенту, який доступний відразу або для дуже критичних для користувача частин інтерфейсу, тоді як асинхронний рендеринг є ідеальним для оптимізації завантаження додаткових даних, модулів або компонентів, які не є необхідними для початкового відображення.

Ключ до ефективного використання цих підходів полягає в аналізі потреб користувача та вимог до продуктивності додатку, а також в постійному тестуванні та оптимізації з метою досягнення найкращого балансу між швидкістю рендерингу, відгуковістю інтерфейсу та загальною якістю користувацького досвіду.

У React можна використовувати як синхронний, так і асинхронний рендеринг для різних частин вашого додатку. Це можливо завдяки механізмам управління станом і рендерингу, які React надає, дозволяючи розробникам оптимізувати продуктивність і користувацький досвід.

Синхронний рендеринг – це стандартний підхід у React, де компоненти рендеряться один за одним у основному потоці виконання. Коли компонент отримує нові props або його state оновлюється, React перерендерює компонент і його дочірні елементи синхронно. Це означає, що інтерфейс користувача блокується до завершення всього процесу рендеринга.

Асинхронний рендеринг у React дозволяє компонентам очікувати завантаження даних або інших ресурсів без блокування інтерфейсу користувача. Це може бути реалізовано за допомогою таких функцій, як React.lazy для лінивої загрузки компонентів і React.Suspense, що дозволяє компонентам «чекати» завантаження необхідного контенту перед їх рендерингом. Конкурентний режим (Concurrent Mode) — це ще одна функція, яка дозволяє React працювати над кількома завданнями асинхронно, покращуючи відгуковість додатку.

Приклади застосування синхроного та аснхроного рендериингу для оптимізації

– Синхронний рендеринг використано для рендеринга основного скелету додатку або критично важливих компонентів, які повинні бути відразу доступні користувачу.

– Асинхронний рендеринг використано для завантаження великих компонентів, модулів, які вимагають додаткових даних з сервера, або для функціоналу, що використовується не відразу після завантаження додатку (наприклад, модальні вікна, додаткові сторінки).

Використовуючи ці методи розумно, можна значно покращили продуктивність додатку і користувацький досвід, зменшуючи час завантаження та відгуковість інтерфейсу. React надає гнучкі інструменти для оптимізації рендерингу, і ефективне їх використання дозволяє досягти високої продуктивності навіть у великих та складних додатках.

Асинхронний рендеринг компоненту в React виконано за допомогою комбінації React.lazy та <Suspense>. Це дозволяє відкласти завантаження компонента до моменту, коли він дійсно потрібен, наприклад, при маршрутизації або відкладеному рендерингу частини інтерфейсу

```
import React, { Suspense } from ‘react’;
const LazyComponent = React.lazy(() => import(‘./LazyComponent’));

function App() {
  return (
    <div>
      <Suspense fallback={<div>Loading...</div>}>
        <LazyComponent />
      </Suspense>
    </div>
  );
}
```

Додаткові зауваження. Асинхронний рендеринг особливо корисний для великих компонентів або бібліотек, що мають значний вплив на час завантаження вашого додатку.

Для використання маршрутизації, застосовано асинхронний рендеринг для компонентів сторінок, щоб кожна сторінка завантажувалася незалежно, покращуючи загальну продуктивність.

1. Оптимізація Контекстно-Залежний Рендерингу

– Пристосування до Умов Користувача: Розробка логіки, що адаптує рендеринг компонентів залежно від контексту (наприклад, швидкості інтернет-з'єднання, типу пристрою), переключаючись між синхронним рендерингом для швидкого з'єднання або потужних пристроїв та асинхронним для повільного з'єднання або менш потужних пристроїв.

Синхронний Рендеринг:

– Адаптація рендерингу компонентів на основі поточного контексту користувача, наприклад, використання різних версій компонентів для різних типів пристроїв або умов мережі, що вимагає синхронного рендерингу для швидкого відображення.

Асинхронний Рендеринг:

– Динамічне включення або виключення компонентів на основі змін у контексті користувача або даних, які вимагають асинхронної обробки для плавного оновлення інтерфейсу.

2. Покращене Управління Даними

Асинхронне Завантаження Даних з Кешуванням: Реалізація стратегій кешування для даних, що завантажуються асинхронно, може значно покращити час реакції додатку, мінімізуючи затримки в рендерингу та зменшуючи навантаження на сервер.

Синхронний Рендеринг:

– Оптимізація доступу до кешованих даних для синхронного оновлення інтерфейсу. Швидкий доступ до кешованих даних дозволяє миттєво оновлювати стан компонентів без затримки.

Асинхронний Рендеринг:

– Використання асинхронних запитів для поповнення кешу та оновлення інтерфейсу на основі нових даних. Асинхронне кешування дозволяє підтримувати інтерфейс актуальним, мінімізуючи час очікування для користувача.

3. Використання Конкурентного Режиму

– Конкурентний Режим React для Покращеної Відгуковості: Експериментування з Конкурентним Режимом для розуміння, як він може допомогти у покращенні відгуковості додатку, дозволяючи React працювати над рендерингом нових оновлень інтерфейсу без переривання поточних взаємодій користувача.

Синхронний Рендеринг:

– В конкурентному режимі React може оптимізувати синхронний рендеринг шляхом пріоритизації взаємодій користувача та важливих оновлень UI, забезпечуючи плавність та швидкість відгуку інтерфейсу.

Асинхронний Рендеринг:

– Дозволяє React працювати над декількома оновленнями інтерфейсу одночасно, використовуючи асинхронність для розподілу завдань рендерингу. Це зменшує блокування основного потоку та покращує загальну відгуковість додатку.

4. Гнучке Управління Станом

– Використання Стейт-Менеджерів: Застосування глобальних стейт-менеджерів (наприклад, Redux або MobX) для ефективного управління станом у великих додатках, що дозволяє краще контролювати процеси оновлення стану та їх вплив на рендеринг компонентів.

Синхронний Рендеринг:

– Оновлення Стану з Кешу: Швидке оновлення стану компонентів за допомогою синхронного читання даних з локального кешу, що забезпечує миттєвий доступ до актуальних даних.

Асинхронний Рендеринг:

– Асинхронне Кешування та Оновлення Даних: Використання асинхронних механізмів для оновлення кешованих даних з сервера, забезпечуючи актуальність даних без затримок в роботі інтерфейсу.

Застосування синхронного та асинхронного рендерингу вимагає ретельного планування та розуміння потреб користувачів та обмежень додатку. Регулярний аналіз продуктивності та користувацького досвіду допомагає виявити оптимальний баланс між швидкістю рендерингу, ефективністю завантаження та загальною якістю взаємодії з додатком.

Висновки. Удосконалення створення кросплатформенних додатків з використанням бібліотеки React та фреймворку React Native є значущим напрямком в дослідженнях з комп'ютерних наук, особливо у контексті WEB- та мобільної розробки. Цей підхід не тільки сприяє підвищенню ефективності розробки, але й відкриває широкі можливості для створення високоякісних, інтерактивних та доступних додатків для широкого спектру користувачів на різноманітних платформах.

Важливо також відзначити роль спільноти розробників у процесі удосконалення та інновацій. Відкритий код та активне спілкування у спільноті дозволяють швидко виявляти та усувати проблеми, розробляти нові функції та покращення, а також сприяють обміну знаннями та досвідом між розробниками. Це створює позитивне середовище для інновацій та росту, сприяє швидкому прогресу технологій та підвищує якість кінцевих продуктів.

Інтегрування синхронного та асинхронного рендерингу сприяє підвищенню ефективності розробки та дає можливість відповідати на швидкозмінні вимоги ринку. Впровадження інноваційних підходів, таких як

конкурентний режим React, глибоке інтегрування з PWA, та розширення можливостей <Suspense>, відкриває нові горизонти для розширення функціональності та доступності додатків.

За результатами аналізу, можна зробити висновок, що подальше дослідження та інтеграція новітніх технологій у процес розробки кросплатформенних додатків буде мати значний вплив на індустрію програмного забезпечення. Розробники, які володіють цими технологіями та оптимізаціями їх застосування, можуть створювати продукти, що не просто відповідають вимогам часу, а формують нові стандарти у сфері мобільної та веб-розробки.

Список використаних джерел:

1. React.memo. Режим доступу: URL: https://uk.legacy.reactjs.org/docs/react-api.html#reactmemo
2. React.lazy. Режим доступу: URL: https://uk.legacy.reactjs.org/docs/code-splitting.html#reactlazy
3. Abramov D. (2015). Getting Started with Redux. Режим доступу URL:https://egghead.io/courses/getting-started-with-redux
4. Kang, J., Kim, Y., & Kim, D. (2017). A study on the implementation of cross-platform mobile application using react-native. Journal of the Korea Academia-Industrial cooperation Society, 18(4), C.155-167
5. UseEffect. Режим доступу: URL: https://uk.legacy.reactjs.org/docs/hooks-reference.html#useeffect

References:

1. React.memo. Access mode: URL: https://uk.legacy.reactjs.org/docs/react-api.html#reactmemo
2. React.lazy. Access mode: URL: https://uk.legacy.reactjs.org/docs/code-splitting.html#reactlazy
3. Abramov D. (2015). Getting Started with Redux. URL access mode: https://egghead.io/courses/getting-started-with-redux
4. Kang, J., Kim, Y., & Kim, D. (2017). A study on the implementation of cross-platform mobile application using react-native.// Journal of the Korea Academia-Industrial cooperation Society, 18(4), P.155-167
5. UseEffect. Access mode: URL: https://uk.legacy.reactjs.org/docs/hooks-reference.html#useeffect