DOI: https://doi.org/10.26642/ten-2023-1(91)-140-145
УДК 004.42

О.В. Войтюк, аспірант
Д.Д. Плечистий, к.т.н., доц.
Державний університет «Житомирська політехніка»

Оптимізація промальовування вебзастосунку на основі об'єктів з глибокою вкладеністю та багатозалежними зв'язками

Стаття присвячена дослідженню застосування методів оптимізації промальовування вебзастосунків з використанням об'єктів з глибокою вкладеністю. Розглядається завдання аналізу користувацького інтерфейсу, який містить складну структуру даних, отриману від серверної частини. Для цього було сформовано спеціальний набір даних на основі аналізу багатьох вебзастосунків, у яких наявна складна система архітектури фронтенд-частини. Продемонстровано, як вона, зі свого боку, слугує основним набором на вхід системи. Показано, що фронтенд-частина розроблена на основі бібліотеки React з використанням таких технологій: JavaScript, HTML, CSS, Redux та допоміжних бібліотек для коректної роботи системи. Додаток структурує та трансформує вхідні параметри у формат, зручний для використання самою системою. З огляду на те, що система орієнтована на використання лише для фронтенд-розробки, обґрунтовано, що серверна частина додатка відсутня. За допомогою браузерних інструментів досліджено проблеми продуктивності існуючих систем, які використовують складні структури даних для промалювання вебдодатка. Доведено, що результатом роботи системи є компонент, який відмальовує оптимізовану систему даних, що має покращені показники продуктивності: швидкість відмалювання, швидкість реакції системи на користувацькі дії, зменшена кількість ітерацій промальовування інтерактивних елементів. Додатковий функціонал містить трансформації і структуризацію вхідних параметрів, унаслідок чого новий набір даних буде мати додаткові параметри та семантичну структуру об'єктів. Надалі планується розробка методу оптимізації і впровадження її у вигляді незалежної бібліотеки, яка, за можливості, замінить процес відмалювання схожих структур даних.

Ключові слова: *промальовування вебзастосунку; фронтенд; React; рендерінг; фреймворк; кеш; рефакторінг.*

Актуальність теми обумовлена тим, що кожен розробник прагне зробити якісний програмний продукт, з дотриманням високих показників продуктивності, зокрема, це: якість обслуговування, доступність, швидкість роботи та дизайн. Дотримуючись цих аспектів, можна розраховувати на утримання користувача всередині будь-якого вебзастосунку. Швидкість промалювання вебелементів – це один з основних факторів швидкодії вебзастосунку, яким не слід нехтувати під час розробки програмного продукту, особливо на перших етапах побудови архітектури проєкту.

Чим менше у вебдодатку елементів інтеракції користувача та системи, тим меншою є необхідність прийняття складних архітектурних рішень. Найчастіше такі додатки мають інформативний характер і показують користувачу сталий вміст даних. Ще однією категорією додатків є системи, які спрямовані на швидке досягнення бажаної цілі користувача, наприклад, чат між користувачами, сервіси запису на прийом, інтернет-замовлення тощо. Зустрічається небагато додатків, в яких фронтенд-частина відмальована не статичними шаблонами, а конфігурована самим користувачем або динамічно змінена системою. Гарним прикладом слугує система налаштування CNC-обладнання, в якій кожна одиниця обладнання має унікальні властивості та технологію обрахунку роботи [8, с. 12].

Надаючи користувачу можливість редагувати усі можливі властивості будь-якої сутності, кожен фронтенд-розробник зустріне проблему «динамічної зміни наповнення додатка», яка несе за собою збільшення часу на обробку та промальовування користувацького інтерфейсу. Здебільшого приклади таких інтерфейсів заздалегідь мають «структурний скелет» на стороні серверної частини, який так само має глибоку вкладеність даних та багатозалежні зв'язки між цими даними.

Швидкість роботи вебзастосунку є однією з основних характеристик якісної роботи сайту, яка безпосередньо впливає на утримання користувача саме на цьому ресурсі. Слабка продуктивність сайту відштовхує користувачів та спонукає знайти інший ресурс. Швидкість роботи сайту залежить від швидкості інтернет-з'єднання, браузера, сервера та клієнтської частини. Правильність побудови архітектури клієнтського додатка та використання методів оптимізації промалювання може значно підвищити продуктивність роботи користувача, адже саме на клієнтській частині відбувається опрацювання запитів та промалювання візуальної частини сайту. Вона так само може бути ускладнена великою кількістю даних, вкладеними формами, прямолінійними зв'язками між різними полями, обрахунками, трансформаціями, роботою зі сховищем даних та анімаціями. З низкою цих ускладнень часто зустрічаються сайти, на яких користувач має справу з конфігураціями. Тому актуальною є наукова задача оптимізації промальовування вебзастосунку на основі об'єктів з глибокою вкладеністю та багатозалежними зв'язками.

Аналіз останніх досліджень та публікацій. У сучасному світі вебтехнологій однією з найпопулярніших бібліотек для створення вебзастосунків є React. З моменту своєї появи React змінив

© О.В. Войтюк, Д.Д. Плечистий, 2023

уявлення розробників інтерфейсу про створення вебдодатків [1]. Деякі методи оптимізації клієнтської частини вебдодатка були описані у публікації В.Г. Фалькевич і Г.Г. Киричек. Авторами було наведено два методи для зменшення кількості рендерів компоненти, а саме: функція життєвого циклу – ShouldComponentUpdate та метод перевірки вхідних значень за допомогою використання PureComponent [2, с. 105]. Однак методи, якими користуються автори, можуть підходити лише для вебдодатків, у яких версія React підтримує класовий підхід до створення компонентів. Нові версії React зупинили підтримку життєвих методів і сконцентрувалися на вдосконалені компонентних підходів, тому нові реалії використання бібліотеки React змушують розробника мігрувати на функціональний підхід до створення компонентів, в якому життєвий цикл компоненти заміняється використанням хуків, які мають підвищену гнучкість використання та більший об'єм покриття задач.

З огляду на проблематику промальовування рекурсивних структур, є недостатня кількість публікацій про розробку методів зменшення кількості часу на промалювання складних користувацьких інтерфейсів. З цієї тематики є матеріали дослідників у галузі веброзробки Уільяма Вана, Ібадехіна Моджеда, Ніко Калєєва, Джеймса Пріста, Пола Льюіса, Кемерона Піттмана, розміщені на блогах LogRocket, Toptal або власних сторінках-проєктах авторів [3–6]. У цих публікаціях значну увагу зосереджено на покращенні швидкості та ефективності вебзастосунків React. Калєєв Н. проаналізував, як оптимізувати CRP (Critical Rendering Path) для найшвидшого рендерінгу і основними пунктами збільшення продуктивності є: компресія, кешування, мініфікація і комбінування CSS файлів [3]. Важливою частиною розробки методів оптимізації є розуміння загальних принципів обміну даними. Ван У. проводить паралелі між оптимізацією та компресію / декомпресією даних, які можуть на 90 % збільшити швидкість оброки даних [4]. Головним показником якісної роботи вебдодатка є стабільні 60 або більше одиниць кадрової частоти, які безпосередньо впливають на утримання користувача на сторінці. Льюіс П. та Піттман К. детально описують сценарій правильного відмалювання додатка, який зможе прибрати візуальні дефекти під час проходження кожної ітерації відмалювання додатка [5]. У роботах І.Моджеда можна відстежувати сучасні практики використання вбудованих методів React, які зменшують час промалювання додатка шляхом додавання функцій перевірки стану вхідних параметрів, як для окремих елементів коду, так і для цілих компонентів [6]. Одним з принципів оптимізації додатка, на думку Є.В. Щербакова та М.Є. Щербакової, є вирішення питання точності вхідних параметрів та змінюваних станів компоненти [7].

Метою статі є розробка методів зменшення загальної кількості часу на промалювання складних користувацьких інтерфейсів вебзастосунків.

Викладення основного матеріалу. Згідно з дослідженням Portent, сайт, який завантажується за одну секунду, має коефіцієнт конверсії в п'ять разів вище, ніж сайт, який завантажується за десять секунд. Тому розробники мають створювати програми з оптимізованою продуктивністю [6]. Для детального дослідження можливості оптимізації промалювання вебзастосунку було проаналізовано сайт замовлення виготовлення тривимірних моделей, а саме приватний кабінет користувача, у якому є можливість налаштовувати параметри виготовлення моделі та налаштування для обладнання (рис. 1).

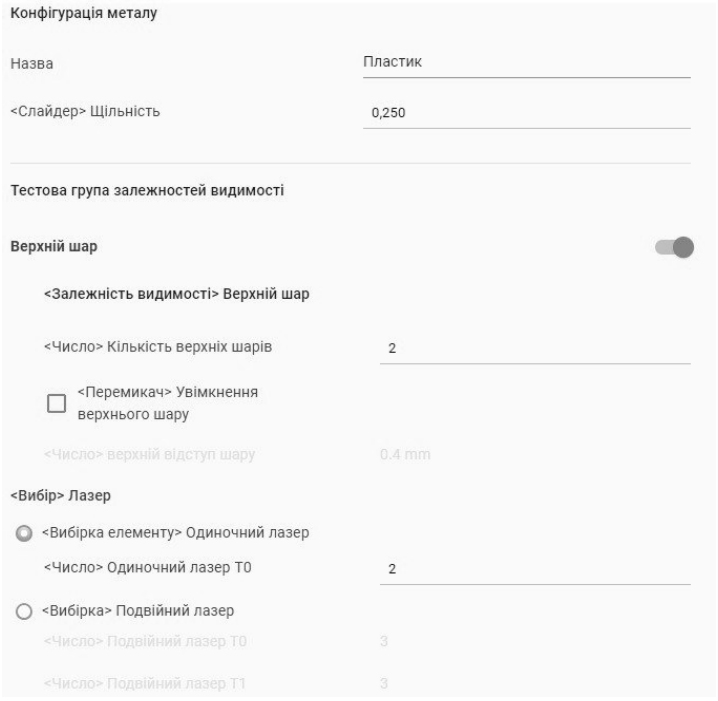


Рис. 1. Сторінка налаштування обладнання

У цьому прикладі можна спостерігати деревовидну структуру фронтенд-архітектури, яка підтверджується відповідним запитам від серверної частини (рис. 2).

Рис. 2. Структура даних, отримана від серверної частини

Маючи готовий приклад, можна переходити до етапу аналізу продуктивності додатка. Основним інструментом аналізу буде слугувати вбудований в браузер Google Chrome – Performance insight. Після проведення аналізу можна помітити велику кількість рендерів компонентів та об'ємні процеси по калькуляції та промальовуванню цих компонентів, що так само свідчить про неоптимізованість системи (рис. 3).

Рис. 3. Загальний аналіз продуктивності сторінки

Додатковим і найпростішим способом перевірки кількості рендерів є виставлення точок повідомлення, однак для цього необхідно знайти в ресурсі сайту код, який відповідає за рендер кінцевих гілок віртуального DOM-дерева. У підсумку (рис. 4) можна зробити висновок, що кількість рендерів для рекурсивного відмалювання усіх компонентів є досить високою, водночас збільшення кількості даних на серверній частині може призвести до значних втрат продуктивності додатку.

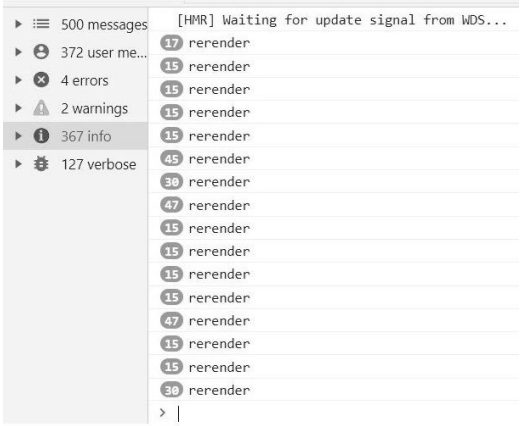
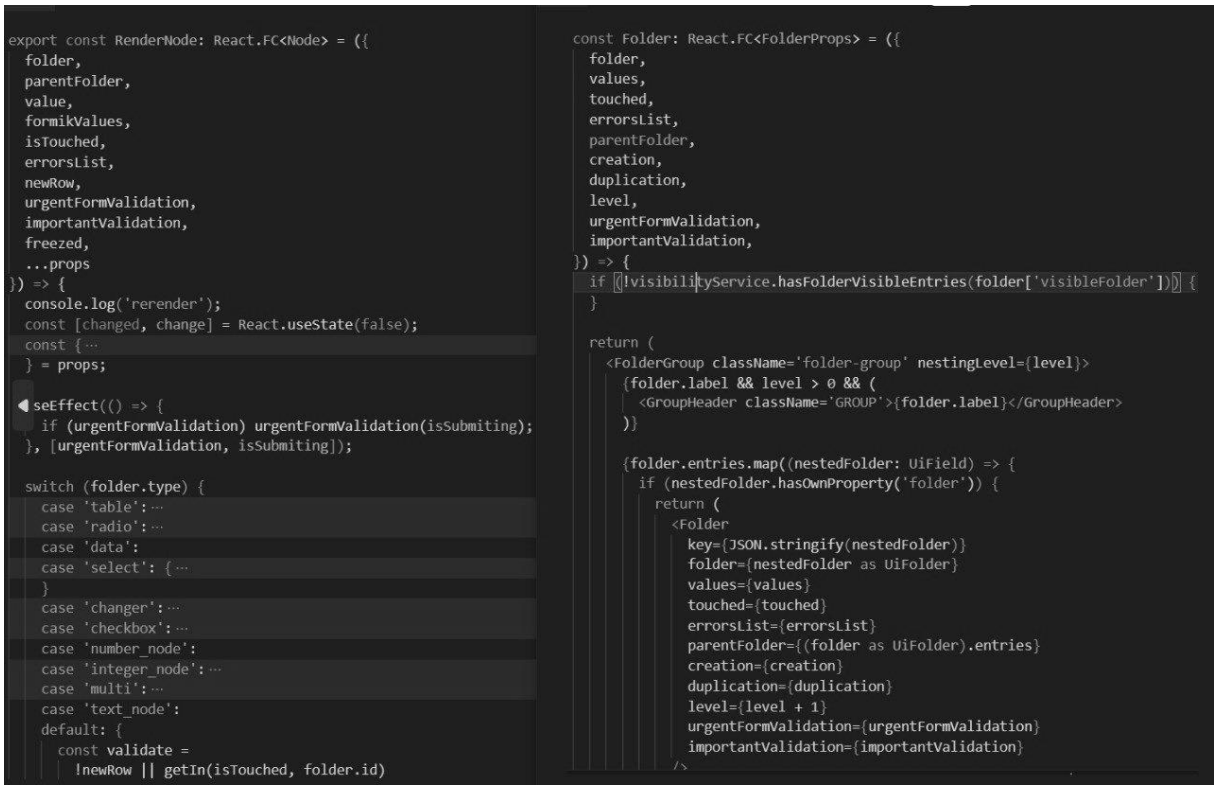


Рис. 4. Аналіз продуктивності за допомогою консолі браузера

Головним елементом аналізу є перегляд рекурсивного методу промальовування всередині коду та усіх дочірніх елементів, які використовуються всередині функції.

На рисунку 5 авторами репрезентовано фрагмент програмного коду з критичними помилками в розробці, які сприяють втраті продуктивності (рис. 5), насамперед це:

- відсутність умов перевірки стану вхідних значень (React.memo);
- відсутність кешування калькуляції і визначення функцій (useMemo, useCallback);
- асинхронність зв'язаних полів;
- велика кількість вхідних параметрів;
- неправильне використання ключового атрибута key;
- навантаження компонентів бізнес-логікою (відсутність розбиття на розумні і прості компоненти);
- наявність «тяжких» змінних, які використовують тільки частину своїх значень;
- недоречна деструктуризація вхідних параметрів.



```
export const RenderNode: React.FC<Node> = ({
  folder,
  parentFolder,
  value,
  formikValues,
  isTouched,
  errorsList,
  newRow,
  urgentFormValidation,
  importantValidation,
  freezed,
  ...props
}) => {
  console.log('rerender');
  const [changed, change] = React.useState(false);
  const {…
  } = props;

  seEffect(() => {
    if (urgentFormValidation) urgentFormValidation(isSubmiting);
  }, [urgentFormValidation, isSubmiting]);

  switch (folder.type) {
    case 'table':…
    case 'radio':…
    case 'data':
    case 'select': {…
    }
    case 'changer':…
    case 'checkbox':…
    case 'number_node':
    case 'integer_node':…
    case 'multi':…
    case 'text_node':
    default: {
      const validate =
        !newRow || getIn(isTouched, folder.id)
```

```
const Folder: React.FC<FolderProps> = ({
  folder,
  values,
  touched,
  errorsList,
  parentFolder,
  creation,
  duplication,
  level,
  urgentFormValidation,
  importantValidation,
}) => {
  if (!visibilityService.hasFolderVisibleEntries(folder['visibleFolder'])) {
  }

  return (
    <FolderGroup className='folder-group' nestingLevel={level}>
      {folder.label && level > 0 && (
        <GroupHeader className='GROUP'>{folder.label}</GroupHeader>
      )}

      {folder.entries.map((nestedFolder: UiField) => {
        if (nestedFolder.hasOwnProperty('folder')) {
          return (
            <Folder
              key={JSON.stringify(nestedFolder)}
              folder={nestedFolder as UiFolder}
              values={values}
              touched={touched}
              errorsList={errorsList}
              parentFolder={(folder as UiFolder).entries}
              creation={creation}
              duplication={duplication}
              level={level + 1}
              urgentFormValidation={urgentFormValidation}
              importantValidation={importantValidation}
            />
```

Рис. 5. Рекурсивне відмалювання кінцевих гілок

Наявність низки таких аргументів є приводом для створення методу оптимізації з покращення продуктивності додатка, шляхом змінення коду та застосування загальних практик і патернів розробки програмного продукту.

Основними критеріями оцінки ефективності методу оптимізації є час промалювання вебелементів та швидкість реакції на інтеракцію користувача, що досягається зменшенням кількості рендерів на сторінці. Вона також мінімізує кількість та розміри стовпців під час перегляду продуктивності сторінки за допомогою інструменту Performance insight.

Шлях оптимізації додатка містить делегацію повноважень на різні компоненті обгортки, які будуть слугувати основою концепції оптимізації коду. Сутність обгорток буде полягати у відстороненні бізнес-логіки від примітивних компонентів та контроль за їх розміщенням і відмалюванням.

Головна обгортка має завдання приймати на вхід невпорядкованою структуру даних і є єдиним джерелом даних для усіх дочірніх обгорток та компонентів. Структура вхідних параметрів може бути як і довільною, так і структурованою. Водночас вона повинна мати такі необхідні дані, як: значення параметрів, кордони можливих значень, вказівники зв'язків між іншими параметрами, заголовки або назви параметрів, списки опцій тощо.

Найважливіша мета головної обгортки – обробити усі вхідні параметри та сформувати нову структуру, яка буде мати низку додаткових параметрів, що будуть слугувати для додаткового контролю за процесом відмалювання і використовуватися у подальшій оптимізації. Під час процесу формування

нової структури даних вся схема деревовидної структури вхідного об'єкта має бути знищена і перероблена у формат звичайного масиву, адже контролювати, слідкувати, змінювати та налагоджувати програмний код набагато легше, використовуючи список параметрів, ніж цей самий список, але з глибокою вкладеністю значень.

Провідна ціль другорядних обгорток – отримати та оперувати даними від головної обгортки, а також комбінувати процеси відмалювання дочірньої компоненти. Обгортка бере на себе роль контролю вхідних даних та прийняття рішення щодо головного питання: «чи потрібне зараз відмалювання елементів?». Сам процес прийняття рішення буде залежати від механізмів перевірки стану вхідних даних, а також додаткових програмних патернів, які зі свого боку будуть запобігати зайвим обчисленням всередині обгортки. Механізм прийняття рішення також спостерігає за іншими обгортками, які мають логічні зв'язки між собою, наприклад, залежно від вибору країни - зміниться вибір міста або залежно від вибору обладнання - зміниться вибір матеріалу, який підтримує це обладнання [8, с. 13].

Можемо виокремити основні стадії життєвого циклу методу оптимізації «обгорток»:

- створення головної обгортки;
- обробка вхідних параметрів (трансформація даних);
- створення додаткових полів у трансформованому масиві;
- створення другорядних обгорток та встановлення усіх правил повторного промалювання;
- створення та налаштування механізму слідкування за зміною залежностей усіх дочірніх компонентів.

Отже, система делегації повноважень засобом створення обгорток дає розробнику можливість легко поєднувати великі обсяги даних у зручній формі та не піклуватися про фінальну продуктивність вебзастосунку. Зменшення числа необхідних рендерів у конкретному вебелементі, що є головним показником якості програмного продукту, сприяє підвищенню ефективності методу оптимізації «обгорток».

Висновки та перспективи подальших досліджень. Метод оптимізації промалювання дозволяє уникнути проблем стосовно продуктивності додатка на етапі відмалювання багаторівневих структур даних та під час їх використання користувачем. Детальний аналіз дозволив нам виявити низку проблем стосовно продуктивності системи. Шляхом використання інструментів розробника та перегляду кодової частини було знайдено вразливості системи, які без додаткового рефакторінгу зможуть ускладнити процес налагодження для розробника та унеможливлювати роботу з додатком для користувача. Підтримка такого коду може збільшити час на пошук та виправлення помилок, а клієнтська частина з легкістю відштовхне вибагливих користувачів. Розробка методу оптимізації та впровадження методу у вигляді незалежної бібліотеки, використання якої зможе замінити процес відмалювання схожих структур даних, буде об'єктом подальших досліджень.

Список використаної літератури:

1. *Chinnathambi K.* Learning react / *K.Chinnathambi*. – Addison-Wesley. – 2019. – 368 p.
2. *Киричек Г.Г.* Оптимізація мережевих систем із використанням front-end технологій / *Г.Г. Киричек, В.Г. Фалькевич* // Вчені записки ТНУ імені В.І. Вернадського. Сер. : Технічні науки. – 2019. – Т. 30 (69), Ч. 1, № 5. – С. 103-108.
3. Critical Rendering Path: What It Is and How to Optimize It [Electronic resource]. – Access mode : https://nitropack.io/blog/post/critical-rendering-path-optimization.
4. Efficient React Components: A Guide to Optimizing React Performance [Electronic resource]. – Access mode : https://www.toptal.com/react/optimizing-react-performance.
5. Browser Rendering Optimization [Electronic resource]. – Access mode : https://james-priest.github.io/udacity-nanodegree-mws/course-notes/browser-rendering-optimization.html.
6. Optimizing performance in a React app [Electronic resource]. – Access mode : https://blog.logrocket.com/optimizing-performance-react-app/.
7. *Щербаков Є.В.* Дослідження ефективності використання бібліотеки React / *Є.В. Щербаков, М.Є. Щербакова* [Електронний ресурс]. – Режим доступу : http://nvdu.snu.edu.ua/wp-content/uploads/2022/08/2022-23-7.pdf.
8. *Войтюк О.В.* Оптимізація промальовування вебзастосунку на основі об'єктів з глибокою вкладеністю та багатозалежними зв'язками / *О.В. Войтюк* // Комп'ютерні технології: інновації, проблеми, рішення : матеріали V Всеукр. наук.-тех. конф., 01-02 грудня. – Житомир : Житомирська політехніка, 2022. – С. 12–13.

References:

1. Chinnathambi, K. (2019), *Learning react*, Addison-Wesley, 368 p.
2. Kyrychek, H.H. and Falkevych, V.H. (2019), «Optymizatsiia merezhevykh system iz vykorystanniam front-end tekhnolohii», *Vcheni zapysky TNU imeni V.I. Vernadskoho*. Ser. *Tekhnichni nauky*, Vol. 30 (69), Part 1, No. 5, pp. 103–108.
3. *Critical Rendering Path: What It Is and How to Optimize It*, [Online], available at: https://nitropack.io/blog/post/critical-rendering-path-optimization

4. *Efficient React Components: A Guide to Optimizing React Performance*, [Online], available at: https://www.toptal.com/react/optimizing-react-performance
5. *Browser Rendering Optimization*, [Online], available at: https://james-priest.github.io/udacity-nanodegree-mws/course-notes/browser-rendering-optimization.html
6. *Optimizing performance in a React app*, [Online], available at: https://blog.logrocket.com/optimizing-performance-react-app/
7. Shcherbakov, Ye.V. and Shcherbakova, M.Ie. (2022), «Doslidzhennia efektyvnosti vykorystannia biblioteky React», [Online], available at: http://nvdu.snu.edu.ua/wp-content/uploads/2022/08/2022-23-7.pdf
8. Voitiuk, O.V. (2022), «Optymizatsiia promalovuvannia vebzastosunku na osnovi obiektiv z hlybokoiu vkladenistiu ta bahatozalezhnymy zviazkamy», *Kompiuterni tekhnolohii: innovatsii, problemy, rishennia*, materialy V Vseukr. nauk.-tekhn. konf., 01-02 hrudnia, Zhytomyrska politekhnika, Zhytomyr, pp. 12–13.

Войтюк Олег Валерійович – аспірант Державного університету «Житомирська політехніка».
https://orcid.org/0000-0003-2254-8977.
Наукові інтереси:
– вебзастосунки;
– інформаційні технології.
E-mail: bubmain25@gmail.com.

Плечистий Дмитро Дмитрович – кандидат технічних наук, доцент, доцент кафедри комп'ютерної інженерії Державного університету «Житомирська політехніка».
https://orcid.org/0000-0002-4803-159X.
Наукові інтереси:
- інформаційні технології.
- комбінаторна оптимізація.
E-mail: kkn_pdd@ztu.edu.ua.

Voitiuk O.V., Plechystyy D.D.

Optimizing the rendering of an object-based web application with deep nesting and many dependencies

The article is devoted to the study of the application of optimization methods for drawing web applications using deeply nested objects. The task of analyzing the user interface, which includes a complex data structure received from the server part, is considered. In this regard, a special data set was formed based on the analysis of many web applications that have a complex front-end architecture system. It is demonstrated how it serves as the main input set of the system. It is shown that the front-end part is developed on the basis of the React library using the following technologies: JavaScript, HTML, CSS, Redux and auxiliary libraries for the correct operation of the system. The application structures and transforms input parameters into a format convenient for use by the system itself. Considering the fact that the system is intended for use only for front-end development, it is justified that the server part of the application is absent. With the help of browser tools, the performance problems of existing systems that use complex data structures to render a web application are investigated. It is proven that the system results in a component that renders an optimized data system with improved performance indicators: speed of rendering, speed of system response to user actions, reduced number of iterations of rendering of interactive elements. Additional functionality includes transformations and structuring of input parameters, as a result, the new data set will have additional parameters and a semantic structure of objects. In the future, it is planned to develop an optimization method and implement it in the form of an independent library, which, if possible, will replace the process of drawing similar data structures.

Keywords: web application drawing; front-end; React; rendering; framework; cache; refactoring.

Стаття надійшла до редакції 18.04.2023.