УДК 004.9
DOI https://doi.org/10.32838/2663-5941/2019.5-1/16

Киричек Г.Г.
Національний університет «Запорізька політехніка»

Фалькевич В.Г.
Національний університет «Запорізька політехніка»

ОПТИМІЗАЦІЯ МЕРЕЖЕВИХ СИСТЕМ ІЗ ВИКОРИСТАННЯМ FRONT-END ТЕХНОЛОГІЙ

У роботі розглянуто методи оптимізації мережевої вебсистеми та процес оптимізації клієнтського інтерфейсу. Методи дослідження базуються на моделюванні системи та схем взаємодії її модулів, а також методах віртуалізації довгих списків і життєвого циклу React ShouldComponentUpdate. Метою роботи є оптимізація мережевої торговельної вебсистеми, розробленої на Laravel. Об'єктом дослідження є процес оптимізації клієнтського інтерфейсу. Предметом – моделі, методи та інструментальні засоби оптимізації клієнтського інтерфейсу. Завдання, які вирішені, є такими: забезпечено зменшення часу очікування під час виконання типових завдань, що не потребують перезавантаження сторінки; оптимізовано перше завантаження сторінки браузером користувача; забезпечено розподілення клієнтської (Front-End) та серверної (Back-End) частин; оптимізовано клієнтську частину, що використовує DOM-дерево; підвищено зручність використання системи клієнтом з урахуванням новітніх апаратних, програмних і соціальних потреб, а також необхідності використання передових front-end технологій. Для досягнення поставленої мети використовується бібліотека React як забезпечення клієнтської частини системи. React використовує декілька розумних методів для мінімізації кількості DOM-операцій, які необхідні для оновлення інтерфейсу користувача, та виконує низку важливих функцій, що реалізовані фреймворком Laravel. Сама система застосовує віртуальний DOM для фонового виконання завдань перед виведенням результатів на сторінку сайту. Для зменшення часу під час роботи з базою даних, особливо під час завантаження серверу баз даних, запропоновано завантаження товарів у невеликій кількості та з попереднім заповнюванням місця під товар анімацією з ефектом довантаження. Функціонал, за який відповідає бібліотека JQuery, реалізовано на JavaScript ES6 та мінімізовано для більш швидкого завантаження.

Ключові слова: *React, front-end, оптимізація, Laravel, сервер, клієнт, JSON, MVC.*

Постановка проблеми. У сучасному світі за постійного зростання конкуренції у сфері інформаційних технологій та росту популярності використання мобільних пристроїв для доступу в мережу Інтернет питання зручності інтерфейсів користувачів є досить актуальним [1]. Тому розвиток інтерактивності і спрощення вебсистем під час користування ними є причиною створення нових методів оптимізації та розроблення зручного й інформативного дизайну мережевих систем. При цьому забезпечення інтуїтивно зрозумілих інтерфейсів надає змогу використовувати системи більш ефективно та знижувати поріг початку взаємодії користувача із системою [2]. Не менш важливим фактором для створення зручного інтерфейсу є час очікування даних, з якими взаємодіє користувач. Чим менший час очікування, тим більш ефективною та зручною для користування є система [2; 3].

Аналіз останніх досліджень і публікацій. Щороку все більше розробників реалізують складні мережеві системи, що доступні користувачам та існують у вебсередовищі. При цьому під час реалізації мережевих вебсистем розроблення розподіляють на Front-End (розроблення інтерфейсу користувача) та Back-End (розроблення серверної частини). Для забезпечення між ними взаємодії використовується інтерфейс, що дає змогу передавати дані з клієнтської частини до серверної та навпаки, незалежно від їх технологій і мов програмування. Найбільш популярними прикладами цих «мостів» є технології JSON та XML [4].

Лідером із розроблення клієнтських частин є мова програмування JavaScript. Після переходу на нову (ES6) версію JavaScript має можливість створення бібліотек і фреймворків під час реалізації клієнтських частин для мережевих вебсистем [1]. Незважаючи на те, що постійно створюються нові бібліотеки та фреймворки для розроблення інтерфейсів користувача, найбільш популярними

технологіями є бібліотеки React та VueJS, а також фреймворк Angular.

Бібліотека React має невеликий розмір, використовує віртуальний DOM (Document Object Mode) для розроблення системи, що дає змогу оптимізувати систему та підвищити швидкість роботи. VueJs, як і React, використовує віртуальний DOM, представляє реактивність і компонентну структуру та фокусується на кореневій бібліотеці, тоді як інші модулі – маршрутизація та управління глобальним станом – належать до інших бібліотек. У React не використовуються html-теги для написання коду. Вона має свій JSX-синтаксис, що допомагає полегшити візуальне розділення коду на елементи структури та функціональні можливості. VueJS надає вибір у використанні JSX-синтаксису або html-тегів. Але VueJS та React під час реалізації повноцінних вебсистем потребують додаткові технології, такі як Flux або Redux, що допомагають управляти станом даних [4].

Аналогом цих бібліотек та основним конкурентом React та VueJS під час реалізації клієнтської частини системи є фреймворк Angular, що використовує мову Type Script. React використовує декілька розумних методів для мінімізації кількості DOM-операцій і робить інтерфейс більш швидким і без використання додаткової оптимізації [5]. Під час аналізу швидкодії компонентів у режимі розроблення можна побачити, як компоненти монтуються, оновлюються та демонтуються під час використання інструментів швидкодії браузерів [6; 7]. На рис. 1 наведено послідовність роботи React. Вертикальна – це послідовність виконання компонентів; горизонтальна – час роботи одного компонента.

Іншим інструментом для аналізу швидкодії React додатка є Profiler (рис. 2). Він дає змогу проводити інтерактивний аналіз кожного компонента. З рисунка видно назви компонентів і час, необхідний для виконання коду [8].

Постановка завдання. Метою статті є оптимізація мережевої торговельної вебсистеми, розробленої на Laravel. Об'єктом дослідження є процес оптимізації клієнтського інтерфейсу. Предметом – моделі, методи та інструментальні засоби оптимізації клієнтського інтерфейсу системи.

Під час вирішення основного завдання дослідження необхідно: забезпечити зменшення часу очікування під час виконання типових завдань, що не потребують перезавантаження сторінки; оптимізувати перше завантаження сторінки браузером користувача; забезпечити розподілення клієнтської (Front-End) та серверної (Back-End) частин; оптимізувати клієнтську частину, що використовує DOM-дерево; підвищити зручність використання системи клієнтом з урахуванням новітніх апаратних, програмних і соціальних потреб, а також необхідності використання передових front-end технологій.

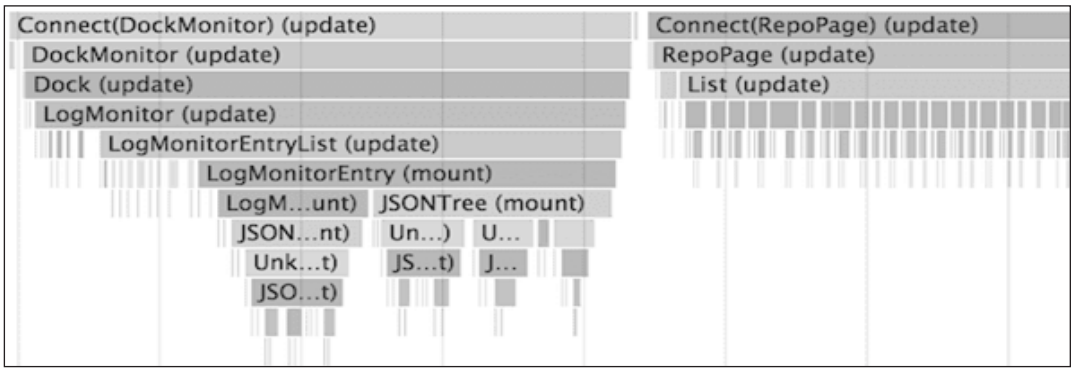

Рис. 1. Аналіз роботи додатка React за кожним компонентом

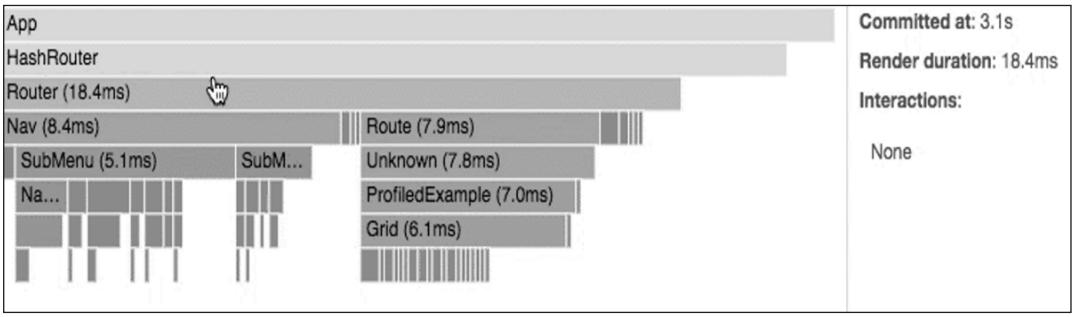

Рис. 2. Аналіз React додатка за допомогою інструмента Profiler

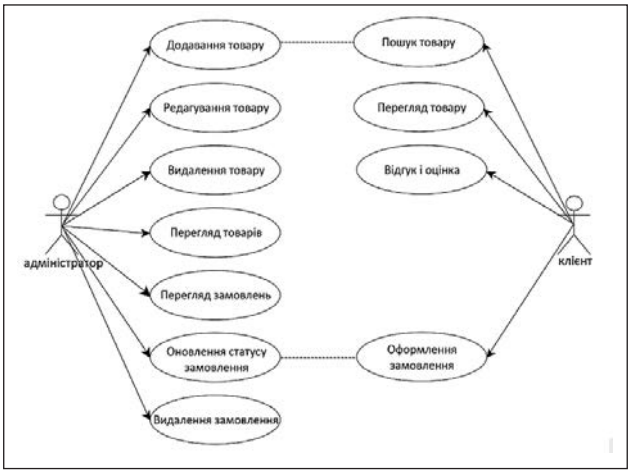

Рис. 3. Діаграма варіантів використання системи

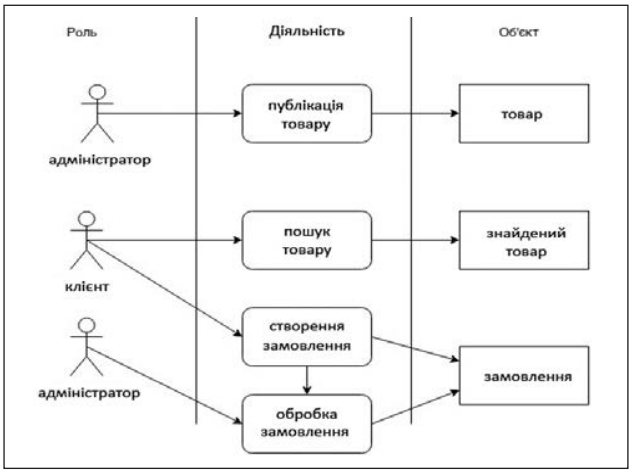

Рис. 4. Інформаційна модель організації системи

Методи дослідження базуються на моделюванні системи та схем взаємодії її модулів, методах віртуалізації довгих списків і життєвого циклу React ShouldComponentUpdate.

Виклад основного матеріалу дослідження. Під час проведення оптимізації реалізованої системи велику увагу приділяємо моделям системи, що наочно демонструють її структуру і поведінку. В цьому випадку маємо систему, яка реалізована на фреймворку Laravel та має клієнтську частину, що складається з файлів шаблонізатора BLADE. Моделі допомагають досягти кращого структурування модулів системи під час проведення її оптимізації. Наведемо діаграму варіантів використання системи (рис. 3) та інформаційну модель її організації (рис. 4).

Технологія проведення бізнес-процесу щодо взаємодії організаційних структур мережевої системи складається з таких етапів: публікація інформації про елементи системи (товар) адміністратором; пошук товарів клієнтом; створення замовлення клієнтом; обробка замовлення адміністратором; відгук та оцінка клієнта. Ця система використовує бібліотеку JQuery, яка працює безпосередньо з DOM-деревом, що збільшує час очікування роботи системи.

Для реалізації нового інтерфейсу користувача, що забезпечить відсутність перезавантажень сторінок під час переходу по сайту, обрано бібліотеку React. Сама система буде застосовувати віртуальний DOM для фонового виконання завдань перед виведенням результатів на сторінку сайту. Функціонал, за який відповідає бібліотека JQuery, реалізуємо на JavaScript ES6 та мінімізуємо для більш швидкого завантаження. Якщо додаток проводить рендеринг довгих рядків, використовуємо метод «Віконного доступу». Цей метод проводить рендеринг тільки невеликої кількості підмножини рядків в один момент часу та дає змогу значно зменшити час, затрачений на повторний рендеринг компонентів і кількість створених DOM-вузлів [7].

Компоненти React додатка мають структуру дерева та кореневий елемент, що містить дочірні компоненти ієрархічної структури. Під час оновлення головного компонента дочірні примусово оновлюються. Це навантажує систему, оскільки не всі компоненти потребують постійного оновлення. Для уникнення цього використовуємо метод життєвого циклу компонента ShouldComponentUpdate з налаштуванням порівняння попередніх і наявних пропсів (Props). Якщо таке порівняння повертає значення ідентичності, то цей компонент не оновлюється. Також для оптимізації компонентів можна застосовувати автоматизацію порівняння пропсів. Методом заборони або дозволу оновлення компонента є використання pureComponents, що імпортується з React-бібліотеки. Для розуміння взаємодії клієнт-сервера та бази даних побудуємо модель розгортання (рис. 5).

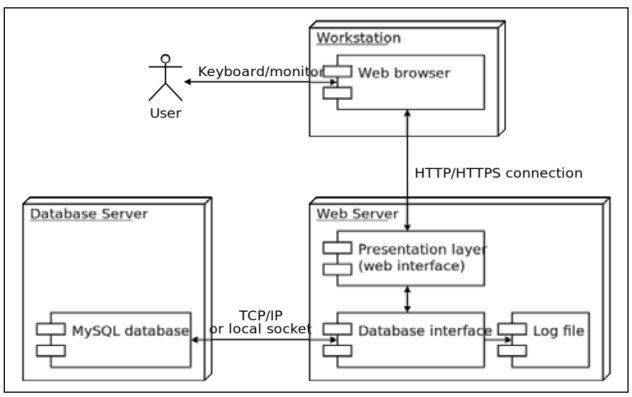

Рис. 5. Модель розгортання

Взаємодія клієнта та системи відбувається так: суб'єкт робить запит через браузер; браузер відправляє запит на сервер по протоколу http або https (шифрування); сервер прослуховує, приймає запити від клієнта та передає на обробку PHP для виконання необхідного скрипта; PHP звертається до бази даних за результатом, використовуючи сокети; база даних надає інформацію, створюється відповідь на сервері та передається до браузера. Модель системи демонструє функціональні можливості додатка, певні послідовності дій суб'єктів та їхню взаємодію. Оптимізація системи, розробленої на Laravel, за допомогою Front-End технологій, не торкається функціональних можливостей і цілісності системи. Вона повинна виконувати свої завдання, використовувати шаблони проєктування і мати ті маршрути, що й до оптимізації. Оптимізація забезпечить зручний інтерфейс і знизить навантаження на сервер.

Перейдемо до включення React. Попередньо встановимо NodeJs, створимо файл package.json, який відповідатиме за необхідні модулі під час подальшої реалізації додатка, та налаштуємо модуль yarn як пакетний менеджер. Далі встановлюємо бібліотеку react, модулі react-dom та babel-preset-react. Вносимо корективи у файл webpack.mix.js: замінюємо рядок коду mix.js('resources/assets/js/app.js','public/js') на mix.react('resources/assets/js/ app.js','public/js'). Тепер маємо точку входу до JavaScript-файлів, при цьому скомпільовані файли розміщені в директорії public/js. Для встановлення залежності запускаємо команду npm install в терміналі.

Після налаштувань у директорії resources/assets/js/component/ створюємо компоненти React. Кожного разу під час реалізації нового компонента він імпортується в початкову точку входу до JavaScript-файлів app.js. Нижче наведено імпорт компонентів до початкової точки входу.

```
require('./bootstrap');          /* Import the Main component */
import Main from './components/Main';
```

Оскільки початкова система використовує архітектурний патерн MVC, необхідно забезпечити збереження архітектурного стилю [9; 10]. Для цього перейдемо до видів (views), що містяться в директорії resources/views. Підключаємо точку входу js-файлів, що містить всі компоненти додатка, до файлу welcome.blade.php. Наведемо код підключення.

```
<!doctype html>
<html lang="{{ app()->getLocale() }}">
<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Laravel React application</title>
<link href="{{mix('css/app.css')}}" rel="stylesheet" type="text/css">
</head>
<body>
<h2 style="text-align: center"> Laravel and React application </h2>
<div id="root"></div>
<script src="{{mix('js/app.js')}}" ></script>
</body>
</html>
```

Мережева система використовує маршрутизацію фреймворка Laravel, тобто після підключення React-бібліотеки та необхідних для подальшого розроблення модулів система перезавантажує сторінки кожного разу під час переходу на інші маршрути. Наразі маємо 16 views, які відповідають за відображення того чи іншого змісту (рис. 6).

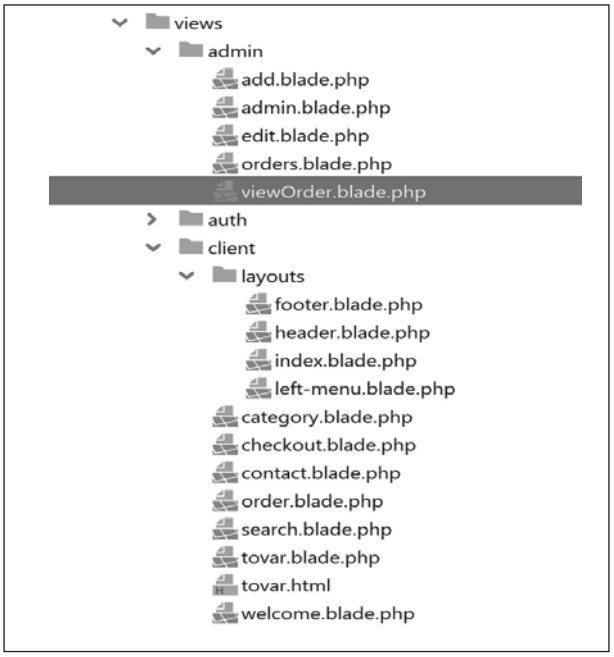

Рис. 6. Laravel views

Для оптимізації перезавантаження кожної сторінки зі схожим змістом під час кожного переходу по сторінках змінімо додаток на Single Page Application. Залишаємо лише два views – для користувача та адміністратора. View користувача – welcome.blade.php, адміністратора – admin.blade.php. Кожен містить загальний скомпільований js-код. Залежно від авторизації на певній сторінці

виконуватимуться певні компоненти. Вносимо зміни до коду контролерів, тепер вони повернуть welcome.blade.php чи admin.blade.php залежно від прав користувача.

Наступним кроком є заміна гіперпосилань на ті, що контролюються JavaScript. Для цього встановимо react-router-dom та замінимо гіперпосилання вигляду на <Route path="" component={назва компонента}/>. Тепер під час натискання посилання фізичного переходу на сторінку не відбувається. Під час тестування система не працює належним чином, оскільки Laravel-маршрути не реагують на віртуальну їх заміну. Для вирішення цієї проблеми реалізуємо необхідні запити через технологію Ajax з використанням XMLHttpRequest та за допомогою функції fetch [11]. Fetch приймає як параметри адресу виконання запиту, тип запиту, заголовки та тіло запиту. Програмний код демонструє використання функції fetch під час створення запиту в React та отримання відповіді.

```
this.state = {
/* Ініціалізація State, який міститиме всі товари, за замовчуванням – пустий масив*/
products: [],            }
componentDidMount() {                    /* fetch API in action */
fetch(`/category/${categoryName}`)
.then(response => {
return response.json();
})
.then(products => {
//Fetched product is stored in the state
this.setState({ products });
});   }
```

Після створеного запиту Laravel-маршрутизація викликає відповідальний контролер, що виконує певні дії. Після вибору категорії react router ініціалізує виконання компонента, що відповідає за її роботу. Далі react-компонент створює асинхронний запит типу GET за адресою /category/ categoryName з передачею параметра назви категорії до маршруту Laravel. Він викликає контролер, який звертається до бази даних і робить запит отримати всі товари цієї категорії. Контролер відповідає на Ajax-запит та передає результат у форматі JSON до методу fetch. Після отримання відповіді рядок JSON декодується в JavaScript об'єкт і записується у State. Під час створення запиту надається відповідь, компонент виконує рендеринг та змінює структуру DOM-дерева, а функція fetch заповнює сторінки даними з бази. Аналогічним чином проводимо оптимізацію інших сторінок і наразі маємо працюючу систему.

У разі оптимізації, яка запропонована, піл час першого візиту до системи постійно необхідно завантажувати бібліотеку React. Використання додаткових модулів також додає розміру системі. Якщо підключення до Інтернету є не досить швидким, з'являється затримка під час відображення сторінки, тому необхідно знизити негативний вплив на інтерфейс користувача. Для цього створюємо прелоадер, який відволікатиме користувача під час першого завантаження системи та матиме анімаційний ефект. Для зменшення часу під час роботи з базою даних, особливо під час завантаження серверу баз даних, пропонуємо завантажувати товари у невеликій кількості та попередньо заповнювати місце під товар анімацією з ефектом довантаження [12].

Висновки. В статті наведено низку методів для оптимізації мережевої системи, знижено навантаження на сервер, розроблено комплекси заходів, що дають змогу покращити інтерфейс користувача, зекономити трафік і кошти на сервер. Оптимізація торговельної системи не торкнулася її функціональних можливостей та цілісності. Збережено останню архітектуру додатка, створено умови для подальшого покращення системи та її розвитку, в повному обсязі виконано поставлені завдання. Під час подальших досліджень планується вдосконалення розроблених методів.

Список літератури:

1. Алексеев Г.В. Современные методы разработки компьютерных систем. Санкт-Петербург : ГИОРД, 2012. 756 с.

2. Ершов А.В., Гуляев Ю.В., Карпов Н.А., Пахомов А.Ю. Интерактивные системы. Москва : ЛИПС, 2014. 596 с.

3. Ahmed T.M. et al. Studying the effectiveness of application performance management (APM) tools for detecting performance regressions for web applications: an experience report. *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 2016. P. 1–12.

4. Сравнение с другими фреймворками. URL: https://ru.vuejs.org/v2/guide/comparison.html.

5. React медленный, React быстрый: оптимизация React-приложения на практике. URL: https://habr.com/ru/post/327364/#measuring.

6. Анализ и оптимизация React-приложений. URL: https://habr.com/ru/company/ruvds/blog/442650/.

7. Оптимизация производительности. URL: https://ru.reactjs.org/docs/optimizing-performance.html.
8. Introducing the React Profiler. URL: https://ru.reactjs.org/blog/2018/09/10/introducing-the-react-profiler.html.
9. Киричек Г.Г., Киричек О.О. Модель оцінки плагіату програмного коду на основі системи контролю версій. *Восточно-Европейский журнал передовых технологий.* 2012. Вып. 2 (2). С. 25–28.
10. Arora A., Sinha M. Web application testing: A review on techniques, tools and state of art //International Journal of Scientific & Engineering Research, 2012. 3(2), P.1-6.
11. Atterer R., Schmidt A. Tracking the interaction of users with AJAX applications for usability testing // Proceedings of the SIGCHI conference on Human factors in computing systems. ACM, 2007. P.1347-1350.
12. Qian Z., Miao H., Zeng H. A practical web testing model for web application testing //2007 Third International IEEE Conference on Signal-Image Technologies and Internet-Based System. IEEE, 2007. P. 434-441.

Kirichek G.G., Falkevych V.G. NETWORK SYSTEMS OPTIMIZATION BY USING FRONT-END TECHNOLOGIES

Network web system optimization and client interface optimization process is described in this article. Research methods are based on systems modeling and its modules, as well as methods of virtual long lists and other React ShouldComponentUpdate cycle. The purpose of this work is to optimize the online trading web system developed by Laravel. The object of the study is the process of optimizing the client interface. The subject is models, methods and tools for optimizing the client interface. Tasks that are solved are as follows: reduced waiting time for typical tasks that do not require a page reload; optimized the first page loading by user's browser; distribution of client (Front-End) and server (Back-End) parts is ensured; optimized client part that is using DOM-tree; increased ease of use of the system by the client taking into account latest hardware, software and social needs, as well as the need to use advanced front end technologies. To achieve this goal, we use the React library to provide the client side of the system. React uses several smart methods to minimize the number of DOM-operations that are required to update the user interface and perform a number of important functions implemented by Laravel. The system uses a virtual DOM, for background tasks execution before displaying results on website page. To reduce the time when working with the database, especially database server operates on heavy load, it is proposed to load products in small quantities and with loading effect placeholders under the product animation. Functionality that is using JQuery library was implemented using JavaScript ES6 and minimized for faster loading.

Key words: *React, front-end, optimization, Laravel, server, client, JSON, MVC.*