Comparison of Redux and React Hooks Methods in Terms of Performance

Daria Pronina [1] and Iryna Kyrychenko [1]

[1] *Kharkiv National University of Radioelectronics, Nauky Ave. 14, Kharkiv, 61166, Ukraine*

Abstract

Data is an essential part of most web applications, and complex applications also have additional state to manage. In this paper the two methods of state management in React applications were considered and analyzed from the performance point of view to provide an additional information when choosing an approach. Confirmed the need to perform careful research when choosing a state manager. The significance of the state management method impact to the application performance, even on a small amount of data, has been proven. Provided recommendations on the studied methods. Redux approach has proven to be not preferred for applications with high performance requirements.

Keywords

Front-end technologies, React, Redux, hooks, performance, single page applications

1. Introduction

The process of creating web applications has evolved and become easier over the years. By operating with various high-level abstractions, developers can design and implement complex systems without having to make any fundamental decisions over and over again.

This rapid development has both advantages and disadvantages. The obvious advantage is that something new appears every day: a new library, a new approach to how to do certain things, which makes the daily routine of a web developer quite diverse. Different frameworks and libraries offer a quick solution to almost any task, introducing restrictions of varying degrees into the program code.

Among the popular libraries for building user interfaces, one of the most flexible is React, with over 15 million weekly downloads [1]. React is a declarative component-based library.

The flexibility of React, in particular, lies in the fact that for a long time it has been positioned as a library that solves only the task of presentation, making it possible to build user interfaces in the form of a component tree.

Creating a simple application with React one may not consider as too complex task as it can be done without any additional libraries.

Thousands of different libraries implement various approaches to managing the state of front-end applications making it hard to decide on the correct approach for a particular project.

At some point Redux library has become very popular. It has its advantages, such as scalability, and it helps make maintaining the application state predictable [2]. But Redux has a lot of disadvantages as well, that have led to the further development of state managers for front-end applications.

And, in addition to this, the hooks were released in React v16.8. Hooks solve a wide variety of seemingly unconnected problems in React and allow to reuse stateful logic without changing the component hierarchy [3]. This led to even more confusion when choosing a state management approach.

One of the other disadvantages of having a wide variety of tool options for solving the same problem is the fact that there are no real standards that can be used when choosing technologies for the development of various applications. This may result in that the chosen technologies are not very

COLINS-2022: 6th International Conference on Computational Linguistics and Intelligent Systems, May 12–13, 2022, Gliwice, Poland
EMAIL: daria.pronina@nure.ua (D. Pronina); iryna.kyrychenko@nure.ua (I. Kyrychenko)
ORCID: 0000-0003-1209-5548 (D. Pronina); 0000-0002-7686-6439 (I. Kyrychenko)

appropriate to use in terms of performance, but simply most common or with which the developer who makes the decision has already had a positive experience.

High-performance websites improve the user experience. By making sites faster, the user experience is improved by speeding up the delivery of the content. Users are most likely to care about the content on the faster sites.

In addition, the performance of the websites impacts not only users, but also website's position in Google search results [4].

2. Related Works

User interface creation has been evolving as well as other components of modern web systems. There are several options for creating and delivering a user interface, each of which has its own disadvantages and advantages over others.

One of the approaches to creating using interfaces is implementing a web application as a single-page application, or SPA. The "Comparison of Front-End Frameworks for Web Applications Development" [5] has highlighted the shift towards single-page applications instead of traditional approach of creating a multi-page application. Three main front-end frameworks were considered in this study and all of them turned out to support single-page applications better or at the same level as multi-page.

The main goal of SPAs is to offer faster transitions on user interactions and to make the web application feel more like a native application that is running locally on the device – in many senses, it is a local application, albeit one that is loaded from the internet rather than from local disk when it is launched [6].

The mentioned above three frameworks, Angular, Vue and React were also studied both separately and in comparison, with each other. For example, in the "DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue, and Svelte" [7] four frameworks were considered and studied from the performance point of view. A lot of metrics were obtained in result of this study: technical metrics, popularity and other.

As a result, React showed the best result in this comparison and is recommended as one of the best choices for front-end application development as it has no discernible weaknesses, and many strengths, won two of the technical benchmarks and performed very satisfactorily in all of them.

Among the advantages of React are the following:

- Virtual DOM;
- wide toolset;
- scalability;
- and many others.

However, the performance of web-application is a compound value, that doesn't depend on the UI library of framework itself. Large web applications operate on a massive amount of data and have a structure of many components, united in a complex page tree, which poses the challenge for developers to maintain the correctness of all data in the application.

Some front-end frameworks solve the problem of data management out of the box, but for long-time third-party solutions were required for such React applications. As a result, various libraries have been created and become a kind of default among developers. The purpose of such libraries is to organize and manage the data flow in front-end applications, in some cases, regardless of whether the application is written in React or not.

The "Comparison of web application state management tools" [8] considered and analyzed four libraries for state management (NgRx, Ngxs, Redux, Vuex). Five criteria were used for the study: code metrics, solution structure, availability of ready-made implementations, community support, and performance testing. In this study a series of tests were performed in order to analyze the performance of each library.

However, some of the considered libraries are framework-dependent and can't be used with any library for creating user interfaces. If considering React, then only one library among studied, can be applied to such applications.

Given that most often when choosing tools for creating a new web application, first of all, a library for creating a user interface is chosen, and only then a state management method in an application and other auxiliary libraries, studies of libraries that can be used with a particular library creating a user interface can be considered more appropriate.

Considering that the amount of data in complex front-end applications can be enormous, the usage of the wrong method of state management can lead to severe performance issues, especially on weaker devices like smartphones, tablets, and older computers.

This makes a study on state management methods from performance point of view indeed essential. The goal of this study is to compare Redux and React hooks methods for state management in React single-page applications and provide the future recommendations based on the obtained results.

3. Preparations

In order to study the impact of different approaches on the performance, a simple front-end application was created with the following functional requirements:

- Application renders list of posts that are retrieved from the third-party API (Application Programming Interface);
- Each post has title, content and associated author (user);
- Each post can be edited and deleted;
- New posts can be created;
- User data retrieved by user id;
- User data can be updated;
- New user can be created;
- User can be deleted.

Additionally, all post data should be stored inside the application, while user data should be retrieved separately for each post.

In this way, implementing the same functional requirements with two different approaches makes it possible to compare them without any assumptions and result data modifications since the experiment will be carried out on the same data.

According to the functional requirements, mentioned above, the following methods for interaction with API should be implemented for post entity:

- Get list of posts;
- Create new post;
- Update existing post;
- Delete the post.

Furthermore, the possibility to access posts state and assign it to new values were implemented.

And for the user entity the required methods are the following:

- Get user by id;
- Create new user;
- Update existing user;
- Delete the user.

The functionality of storing current user data, as well as possibility to retrieve and update it, was implemented.

Presentation part of the application once implemented with React, remains unchanged during the study, while the state management is being implemented with React Hooks and Redux approaches separately.

4. Implementing data management with React Hooks

For React Hooks implementation no other dependencies are required except the React library itself with the version starting from 16.8.

React Hooks approach provides an opportunity to implement all the logic directly inside components, whether it is a basic operation with state, interaction with browser APIs or network request.

This straightforward approach at scale brings a lot of duplicated code, as well as constrains logic to specific components, resulting in the tightly coupled code. That increases the complexity of code supporting and lowers the degree of code reusability.

Although this can be avoided by creating and using custom hooks. Custom hook is a function, that uses bult-in React Hooks in order to provide a more complex functionality. In such way the encapsulation of all logic, related to API or other state manipulations, can be achieved.

Built-in React Hooks that were used:

- useMemo;
- useCallback;
- useEffect;
- useState.

These hooks are designed for storing memorized (that stand for memory optimization) values, callbacks and simple components state data as well as performing side effects.

Memorization has its cost and should be used carefully, as it can affect the performance of the application and make it even worse instead of bringing any optimizations.

This is useful when transferring callback functions to optimized child components that rely on equality of reference to avoid unnecessary rendering.

In addition to the built-in hooks, a custom hook useAPI was implemented. In this hook all methods for http requests are encapsulated, as well as API URL.

The methods provided with the custom hook:

- getAll;
- getById;
- create;
- update;
- delete.

The API, used for the research, is RESTful, and, therefore, the API calls can be easily reused by only providing the entity name in configuration and a callback.

5. Implementing data management with Redux

Redux is a third-party library for state management that is used in front-end applications. The main concept behind Redux is that the entire state of an application is stored in one central location. Each component of an application can has direct access to the state of the application without having to send props down to child components or using callback functions to send data back up to a parent [6].

The global state of the application (store) is read-only. In order to make changes there, it is necessary to dispatch a corresponding action.

Changes in the global state occur through pure functions (this means that these functions should not have any side effects).

With Redux, we operate with concepts such as actions and reducers. By default, actions are synchronous and not suited for such tasks as API calls, so an additional library was installed along with redux and react-redux libraries.

The official React bindings for Redux are implemented and provided with react-redux library. Using methods provided in this library, React components can be attached to the application store or to some part of it, as well as the possibility of dispatching actions is provided.

Building an architecture for complex application with Redux is not an overly difficult task, as the main concepts of it already separate different responsibilities. Although the amount of boilerplate, introduced by Redux, one may call overly excessive.

Actions in Redux are plain objects, that have type and payload, however this is modified with the redux-thunk library, so the actions are transformed into asynchronous functions.

Reducer is a pure function (that means that there should not be any side-effects within reducers), that accepts the current state and the dispatched action, then process it and return the next state.

All reducers, as well as additional middlewares, are combined into a store, that is passed to the React application by the special component-provider.

The use of Redux involves the creation of all structures that implement this approach.

The data flow in Redux in shown in Figure 1.

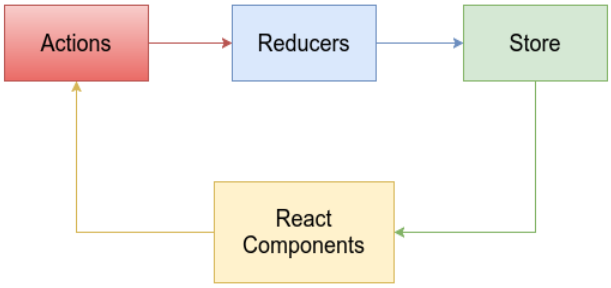

Figure 1: Redux data flow

The same methods as in case with Hooks were implemented using Redux.

6. Experiment

The influence of the chosen state management approach can be observed both on the output code (in the case of a front-end application, this is the size of the JavaScript file) and on measurements in the directly running application.

It is important to think about the performance on all stage of the application development, not just when the critical moment comes. It can be said that software evaluation at the design stage has a high practical application, since allows you to perform a software assessment before it starts, which in turn allows you to take into account most of the possible risks of the project and the development stages. In turn, this allows you to compare what costs are needed and what the future cost of the project [7].

The performance of your web application can be degraded very easily by using wrong libraries not only for the state management. And the metrics and their boundary values are usually determined individually for each project if there are any.

Google has created a model called Web Vitals for providing development guidelines to developers in the process of designing and building web applications, which are extremely important for providing an adequate user experience on the web. The indicators defined in this way are an empirical measure of the actual user experience that Google analyzes and which are based on the key needs of users: page loading speed, application interactivity, stability, content experience, etc. [8]

Web Vitals is a useful tool for evaluating web application's performance.

Core Web Vitals contains the following metrics:

- Largest Contentful Paint;
- First Input Delay;
- Cumulative Layout Shift.

Besides the Core Web Vitals metrics there are also other metrics for different stage of interaction between the user and the page. Though most of those metrics is not influenced by the state management approach, but with other parts of the front-end-related part of the web application [9]. Therefore, it was decided not to use Web Vitals directly.

Nevertheless, the metrics, created by Google, were taken into account. It was assumed that the most impact of the state manager choice can be observed on the build size and while user interacts with the page.

The bigger the build size, the more it takes for user device to download all the application files and launch it. Time, that it took for page to load and become interactive, is a very important metric as it affects the potential user significantly.

As for metrics related to user interactions, it is unnecessary to prove their importance, as they are basically the core of user experience. And, additionally, wrong state management may lead to confusion and business losses.

Taking in the account everything mentioned above, the following metrics were defined for the comparison:

- Percentage delta in amount of code needed for implementing base methods for the first entity in the project;

- Percentage delta in amount of code needed for implementing base methods for the entity in project with other entities already implemented;
- Percentage of the reused code;
- Assumed further code amount growth rate for the new entities;
- Delta in size of state management code for Hooks and Redux versions in production bundle;
- Delta of memory used for storing the same amount of the application data;
- Delta of time for state update operation.

7. Code size comparison results

Measurements regarding the code size of the project are shown in Table 1.

Table 1
Code size measurements

№	Metric	Size measurements in Hooks version	Size measurements in Redux version
1	First entity methods	2.01 KB	3.79 KB
2	Second entity methods	742 B	2.34 KB
3	Shared code between entities	1.02 KB	1.44 KB
4	Total code for both entities	2.75 KB	6.19 KB
5	State management code in production bundle	1.26 KB	16.49 KB

The following formula is used for calculation of the percentage delta in amount of code needed for implementing base methods for one entity both in the new project and in project with implemented other entities:

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

where V_1 is the size of code that was developed for implementing methods of the first entity in KB and V_2 is the size of code for the second entity respectively.

The results of calculations are presented in Table 2.

Table 2
Delta of base methods code

Metric	Value for Hooks version	Value for Redux version
$V_{\Delta bm}$	63.95%	32.26%

The importance of the amount of code that can be reused grows in proportion to the number of entities and, therefore, is worth calculating. The following formula is used for calculation of the reused (shared) code percentage:

$$V_r = \frac{V_1}{V_2} * 100\%, \quad (2)$$

where V_1 is the size of code that is used by both entities in KB and V_2 is the total size of code for both entities in KB.

The results of calculations are presented in Table 3.

Table 3
Percentage of the reused code for both entities

Metric	Value for Hooks version	Value for Redux version
V_r	37.09%	23.26%

Assuming that the following entities will take the same amount of code to be implemented as the second entity, the growth rate can be calculated by the following formula:

$$V_g = \frac{V_1 - V_2}{V_2} * 100\%, \quad (3)$$

where V_1 is the total size of code for both entities in KB and V_2 is the size of code for the first entity.

The results of calculations are presented in Table 4.

Table 4

New entities code growth rate

Metric	Value for Hooks version	Value for Redux version
V_r	36.82%	63.32%

8. Application performance comparison results

For measuring performance of the running web app Chrome Dev Tools may be used. Chrome DevTools is an interactive dashboard, that has a lot of different tools for web developers included, and it is bult-in in the Chrome browser.

The example of Performance tab look is shown in Figure 2.

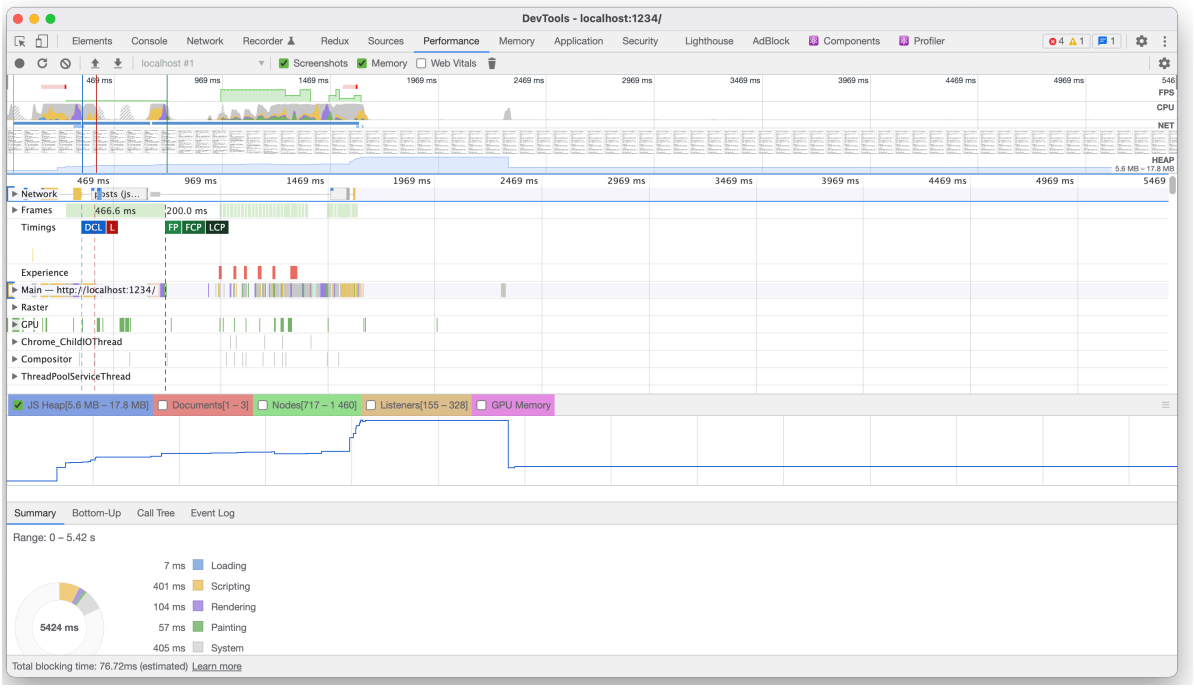

Figure 2: Performance tab

Among the presented tabs, the one called Performance may provide a lot of important information and shed some light on how it actually works under the hood.

While storing application data with Redux or Hooks, the memory of user's device is consumed. As there are many ways for storing optimization with different data structure and approaches, the required size for the same data in Redux and Hooks may not be equal.

The amount of memory was measured for different size of data stored in an array 10 times and after that the mean value was calculated. The results of measurements are presented in Table 5.

Table 5

Memory usage

Data size (length of array)	Mean value for Hooks version	Mean value for Redux version
130	8.5 MB	9.1 MB

150	12 MB	17.1 MB
200	16.6 MB	27.1 MB

Applications are designed to be interacted with by users, therefore the responsiveness is another important metric in terms of performance. One of the possible interactions in the application in question is creation of the new post, that causes a http request to the server and state update as well, that causes the page re-rendering. Google Chrome Dev Tools allows us to measure the exact time for this interaction.

Figure 3 shows the example measurement for creation of the new post action.

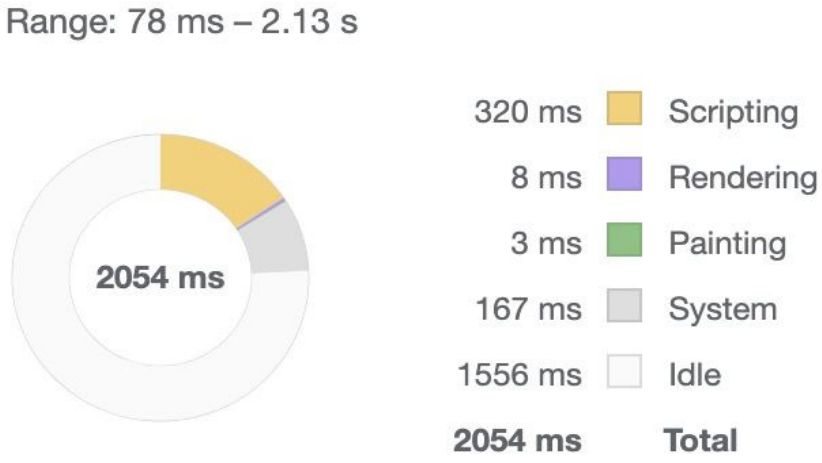

Figure 3: Recorded performance on updating state

The actual consumption time to update state is calculated by using the following formula:

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

where T_t is total consumption time and T_i is idle time [10].

The results of measurements are presented in Table 6.

Table 6
Time for state update operation

Mean value for Hooks version	Mean value for Redux version
102ms	325ms

The average value was taken by measuring the interaction 10 times and, afterwards, calculating its mean.

9. Discussions

The two methods of state management in React single page applications were compared from the performance point of view. Several metrics were considered and measured in order to obtain a full information of the selected methods.

As can be seen from the results, the reduce of code size for implementing the new entity methods in Redux version equals to almost a half of the value for the Hooks version, and that is the result of the fact that using Redux requires a lot of boilerplate code in order to maintain its architectural approach in state flow.

It is an important metric as complex web applications operate a large amount of entities, so the cost of new entities introduction should be considered when developing an architecture for the application.

The study results show that in the Hooks version the percentage of the reused code is higher. This may indicate that in long-term perspective the maintenance of this code will be easier, comparing to

Redux, although one definitely may increase this value in Redux version by using various libraries and, therefore, enlarging the final bundle size.

On the other hand, the code reusing may result in a high coupling parameter, which is not a good option in large projects. Nevertheless, the React Hooks method allows a significant flexibility when it comes to code organization, which is a major advantage on Redux.

The results of new entities growth rate metric are quite predictable after calculating all previous metrics. Redux approach showed worse results for each metric and the main reasons for this are the following:

- Introducing such concepts as actions, reducers, middlewares;
- A lot of boilerplate.

The boilerplate may be reduced by using third-party libraries. However, the size of state management code in production bundle for Redux already shows a large value in comparison to Hooks approach (can be seen in the Table 1). The reason is that Redux itself is a third-party library, and, moreover, this can be caused because of the required additional libraries, such as:

- react-redux;
- redux-thunk.

These libraries are essential for Redux library to work with React library and are used in most of the projects, although the redux-thunk library has other alternatives.

There are also another popular libraries that were designed and developed to work with Redux in order to achieve different purposes and optimisations. However, it is not known for sure whether these optimisations are related to application performance or only to developer experience, which is also a valuable metric in project development process.

The difference of the memory usage wasn't noticeable on small amounts of data (less than 120 items in an array), but it grows faster in Redux version.

Based on the data obtained, it can be concluded that simple operations in Redux approach take more time than when using React Hooks.

There is always a place for additional optimisation, but one should take into account the amount of additional work and research needed in order to use the one or the other method.

The results of the "Comparison of web application state management tools" showed that Redux is one of the fastest among other state managers that were considered in that study. As the results of the current study shows that Redux is slower from the performance point of view than React Hooks, it can be concluded that Ngrx and Ngxs methods, that are suitable for Angular framework, are not optimized from the performance point of view.

Considering the fact that React is one of the fastest libraries for user interface creation, that was proved in the according study "DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue, and Svelte", it becomes obvious that in case of high-performance necessity React and React Hooks should become a number one choice for single-page application development.

10.Conclusions

In this study two methods of state management in React single-page applications were considered from a performance point of view: Redux and React Hooks.

A series of tests and measurements were carried out to get a complete picture of how each of the approaches affects performance.

Redux approach requires a lot of boilerplates in order to implement the correct data flow and follow the main guidelines and concepts. In addition to that, Redux also requires the installation of the react-redux library, as well as a library for creating asynchronous actions, in order to make http requests possible. That resulted in worse performance compared to React Hooks approach.

The study showed that creating the data flow logic is more complex in Redux and takes more code. Also, it took more memory for Redux to store the same amount of data in comparison to React Hooks approach.

The basic state changing operation consumed more time in Redux approach.

To sum all up, it can be concluded, that React Hooks approach is more optimized from the performance point of view than the Redux approach.

As the study was based on a simple application created solely to compare two approaches in performing the same tasks, it is not known for certain whether the same results will be obtained on a more complex application.

However, the obtained results showed the significant delta in used resources even on small amount of data depending on the state management approach. This proves that the process of choosing a state manager must be approached responsibly, because both its shortcomings and the replacement process can be very costly.

In case of high importance of application performance, it is not recommended to choose Redux as a state management library in React single-page applications.

Further research is required to confirm the results obtained and the conclusions drawn, or to refute them. Complex web-applications with several entities and additional application-wide state features, such as notifications and authorization, should be considered for further research. Other methods of state management should be compared in order to provide a clearer picture of the current state of state management in terms of performance.

11.References

[1] Npmjs.Com, React library in the npm Registry, 2022. URL: https://www.npmjs.com/package/react.

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

[3] Dan Abramov, React v16.8: The One With Hooks, 2019. URL: https://reactjs.org/blog/2019/02/06/react-v16.8.0.html.

[4] Jeremy Wagner, Web Performance in Action: Building Fast Web Pages, Simon and Schuster, New York, NY, 2016.

[5] Marin Kaluža, Krešimir Troskot, Bernard Vukelić, Comparison of Front-End Frameworks for Web Applications Development, Zbornik Veleučilišta u Rijeci, Vol. 6 No. 1, 2018. doi: 10.31784/zvr.6.1.19.

[6] Appcheck-Ng.Com, Single Page Applications, 2020. URL: https://appcheck-ng.com/single-page-applications/.

[7] Mattias Levlin, DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue, and Svelte, Master's Thesis, Åbo Akademi University, Turku, Finland, 2020.

[8] Kacper Szymanek, Beata Pańczyk, Comparison of web application state management tools, in: Journal of Computer Sciences Institute, 20, 2021, pp. 183-188. doi: 10.35784/jcsi.2675.

[9] Sophia Shoemaker, Mark Erikson, Redux: Why It's Good For You, 2017. URL: https://www.newline.co/fullstack-react/articles/redux-with-mark-erikson/.

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

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

[12] Smelyakov K., Smelyakov S., Chupryna A. Advances in Spatio-Temporal Segmentation of Visual Data. Chapter 1. Adaptive Edge Detection Models and Algorithms. Series Studies in Computational Intelligence (SCI), Vol. 876. – Publisher Springer, Cham, 2020. – pp. 1-51. doi: 10.1007/978-3-030-35480-0.

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