Implementation of Web System Optimization Method

Galina Kirichek[1[0000-0002-0405-7122]], Stepan Skrupsky[1[0000-0002-9437-9095]], Mariia Tiahunova[1[0000-0002-9166-5897]] and Artur Timenko[1[0000-0002-7871-4543]]

[1]National University "Zaporizhzhia Polytechnic", 64 Zhukovsky str., Zaporizhzhia, 69063, Ukraine

kirgal08@gmail.com, sskrupsky@gmail.com, mary.tyagunova@gmail.com, timenko.artur@gmail.com

Abstract. The method of optimization of the network web system client interface has been considered in the paper. The system model demonstrates the specific sequences of subject's actions and their interactions. The proposed work uses modern technologies and techniques, including React library and its features to create a "smart" client. It can partially perform the functions of server logic, thereby providing less server load and improving the interactive user interface. The research methods are based on interaction schemes modeling of the system and its modules, the methods of virtualization of long lists and the shouldComponentUpdate React lifecycle method. Optimization with Front-End technologies does not affect the functionality and integrity of the system.

Keywords: optimization, Front-End, React, Laravel, client, interface

1 Introduction

Nowadays, with ever-increasing competition in information technology and the increasing popularity of mobile devices for accessing to Internet, the issue of user-friendliness is relevant [1, 2]. Therefore, the development of interactivity and simplification of web systems, when using them, is the reason of the creation of new optimization methods and development of user-friendly and informative design of network systems. By providing intuitive interfaces, it allows you to use systems more efficiently and reduce the threshold for the onset of user interaction with the system [3]. Equally important, to create a user-friendly interface, is the latency of the data that the user interacts with. The shorter the waiting time, the more efficient and user- acceptable is the system [4, 5].

Every year more developers implement complicated network systems, which available for consumers and exist in web environment. When network web systems are in process of implementation, the development is divided to Front-End (development of user interface) and Back End (development of server side). For ensuring a cooperation among them an interface is used which gives an opportunity to transfer data from the client side to the server one and vice versa, independently of programming languages.

The most popular examples of these “bridges” are technologies JSON and XML [6, 7]. The leader in development of client’s applications is a programming language JavaScript. After transition to the new (ES6) version, JavaScript has an opportunity to create libraries and frameworks when client side parts are implemented for network web systems [1]. Although new libraries and frameworks are being built to develop user interfaces, the most popular technologies are React and VueJS libraries and Angular framework.

A framework Angular, which uses Type Script language, is an analog and the main competitor of these libraries in implementation the client side of the system. React uses a few smart methods to minimize numbers of DOM operations and makes an interface faster without using additional optimization [8]. During component performance analysis, in development mode, you can see how components are mounted, updated, and unmounted by using browser performance tools [9, 10].

2 Problem statement

In solving the main task of the research it is necessary to provide a reduction of the waiting time when performing typical tasks that do not require a page reload; optimize the first page load of a user's browser; ensure the distribution of client (Front-End) and server (Back End) parts; optimize the client part using the DOM tree; increase the usability of the system usage by the client taking into account the latest hardware, software and social needs, as well as the need to use advanced Front-End technologies.

During optimization of the implemented system we pay much attention on the models of the system, which demonstrate its structure and behavior. In this case we have the system that has been implemented with using the framework Laravel and have the client side, which consists of Blade templates. UML models help to structure the modules of the system while performing its optimization. Here is use-case diagram for the system (Fig.1) and information model of its organization (Fig.2).

Fig. 1. Use-case diagram

A technology of maintaining a process relatively interacting with organizational structures of the network system consists of next stages: information publishing about items of the system by administrator; searching the information by a client; creating an order by client; processing the order by administrator; client's feedback and assessment. This system uses the library jQuery, which works with a DOM tree directly and increases the waiting time of working system.

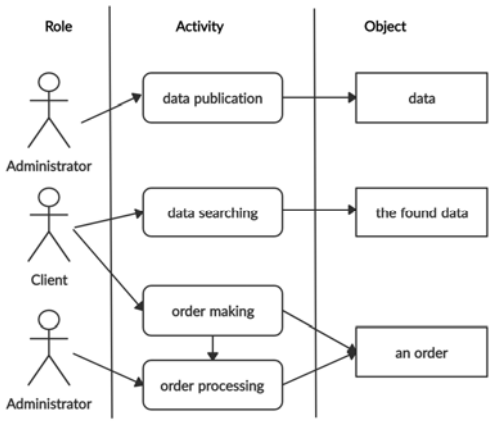

Fig. 2. Informational model of the system organization

The goal of this work is an optimization of the network web system, which has been developed by using Laravel. An object of research is client interface optimization process. Subject is models, methods and optimization tools of the system client interface.

The methods of research are based on modeling of the system and modules interaction schemes, methods of long lists virtualization and shouldComponentUpdate React lifecycle method.

3 Modeling of system

Features which were implemented by using the jQuery library will have been written by using JavaScript ES6 and will have been minimized for faster downloading. If the application makes rendering of long rows, we use a method of “Window’s accessing”. This method makes rendering only a small amount of a subset of rows during one timeframe and allows considerably reducing time which is needed for repeating rendering of components and numbers of creating DOM knots [10].

React components of application have the structure in a shape of tree and the root element, which includes children components of hierarchical structure. If the main component is updated, children components forcibly are updated too. It loads the system, as not all components have to be updated. To avoid this, we use the Should-

ComponentUpdate component lifecycle method to configure the comparison of previous and existing Props.

If this comparing return true, then the current component does not have to be updated. Also, for optimization of components it is possible to apply the automation of the comparing props by the method of prohibition or permissions of updating the component with using of the pureComponents, which are imported from the React library. For understanding of interaction client-server and a database we will build deployment model (Fig.3).

Optimizing the Laravel-developed system with Front-End technologies does not affect the functionality and integrity of the system. System must fulfill its tasks, use design patterns and have the same routes as before optimization. Optimization will provide a user-friendly interface and reduce server load.

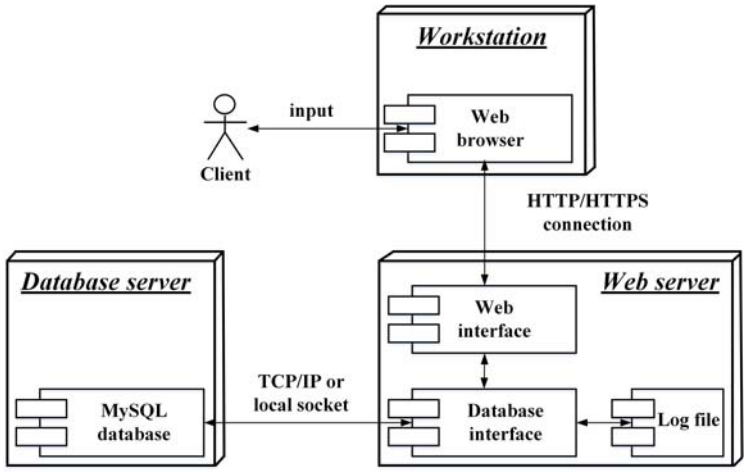

Fig. 3. Deployment model

Let's pass to inclusion of React. Install NodeJs, create the file "package.json", which responsible for modules which are needed for future development of the application and set up the module Yarn as a package manager. Next step is installing of the React library, modules react-dom and babel-preset-react. Make corrections in the file webpack.mix.js: replace a row of code mix.js ('resources/assets/js/app.js', 'public/js') to mix.react('resources/assets/js/ app.js','public/js').

There is a point of entering to JavaScript files, compiled files are placed in the directory public/js. For installing dependencies execute the command "npm install" in the command line.

After settings were installed, create React components in the directory resources/assets/js/component/. Every time a component is implemented, it is imported to the entry point to the JavaScript file app.js. Import of components to the entry point is given below.

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

```
import Main from ‘./components/Main’;
```

As the system uses the design pattern MVC, it is necessary to ensure its usage [11, 12].

Let's pass to the views, which are placed in the directory resources/views of project. Import an entry point of js files, in which all application components are defined, to the file “welcome.blade.php”. The fragment of code is given below.

```
<!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>
```

The network system uses routs of the Laravel framework, so after the React and modules that are necessary for future development are connected, the system reloads pages every time when the user uses other routes [13, 14].

4 Optimization of system

After examining the precedents of roles and sequence diagrams, we modeled the principle of operation of the system, taking into account the Laravel framework and its architectural style (Fig.4). Zone "A" is the client, "B" is the server, "C" is the database.

By default, the Laravel framework is implemented using the Blade Template. This template helps simplify system partitioning into client and server parts. In this case, a conditional distribution is implied, since the view as well as the controller and model are located on the server and can theoretically fulfill the logic of the server.

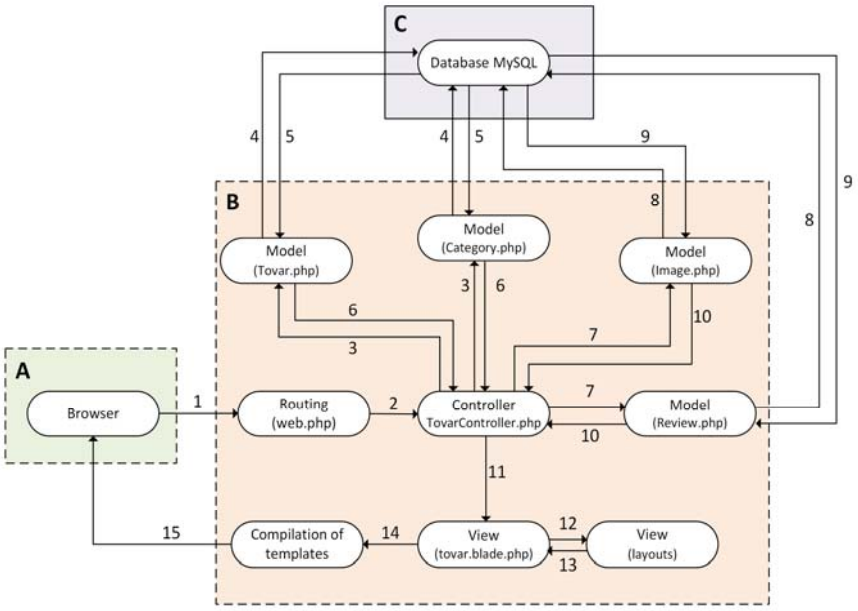

Fig. 4. In-app request processing algorithm

Fig. 4 shows a model of a system based on Laravel architectural style that involves the use of an MVC pattern (model, view and controller). Consider a modified model that will use the same pattern, but with the inclusion of Front-End technology to client. Now it is possible to speak not only about the formal division of the system into client and server (as in Fig. 4), but also about the physical one, since now the view (template) is fully browser-based and has its own logic.

Consider changes in the algorithm of the application. Fig.5 shows the changes in operation of system:

- User requests are managed by the React library (A).
- User request is tracked by JavaScript code and transmitted to controller (1).
- Controller is connected to model (2).
- Model creates a request using JSON technology (3).
- Server has a special intermediary – the API that meets requests from the client, decodes the string in JSON format, parses the essence of the request, checks the headers for identification of the client who sent them.
- API creates the required request and sends it to Laravel (4).
- Controller Laravel TovarController framework transfers the data to the API (14).
- API encodes the data received from the controller and returns them to the client part (15).
- Client receives the data and adapts it to the user's display (16, 17).

The user browser only takes steps to display the content of the page on the screen. However, modern browsers contain enough resources to perform more complex tasks.

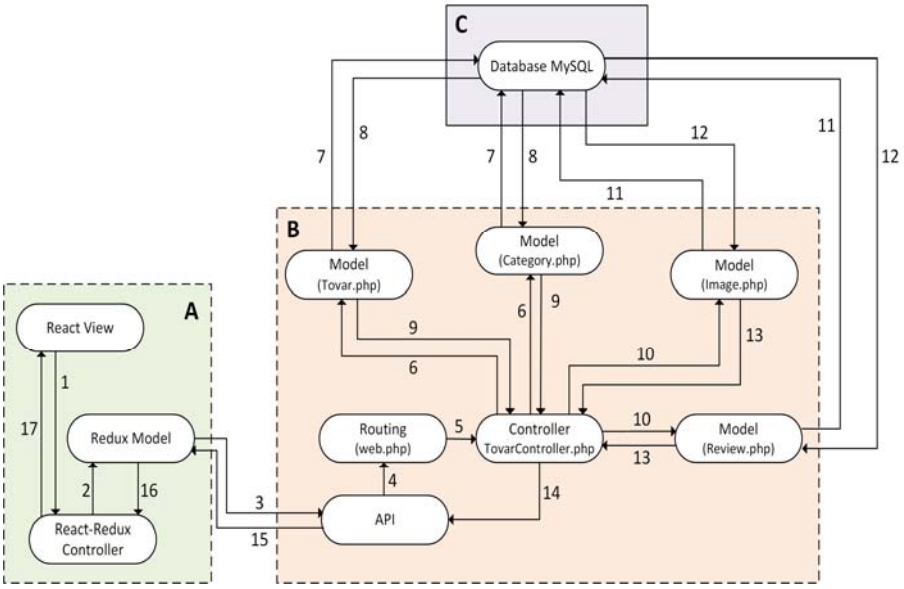

Fig. 5. Request processing on the system with React library

Having a system model using the React library and algorithms to change client-server interactions, let's move on to optimization of the system. General stages are the following:

- Measurement of page load time, external resources in a non-optimized system.
- Measurement of time of script execution.
- Implementation of the react library in the system.
- Configuring load balancing between server clusters using JavaScript code.
- Replacing the jQuery library with native JavaScript code.
- Setting the sequence of loading of third-party resources and basic JavaScript of the system.
- Setting up parallel load of resources.
- Providing interactive download.
- Minimizing js, css, html files. Image optimization.
- Setting of data caching. Setting up Gzip data archiving.
- Measurement of page load time, external resources after system optimization.
- Comparing optimized and non-optimized systems with slower internet connections.

The optimized system with Front-End technologies involves a large amount of JavaScript code, React libraries, which in turn may also have third-party modules, can primarily affect the first time the client is loaded and, moreover, the user's expectations, so let's provide interactive download.

Based on the classification above, the first thing that should appear in the client's browser is the information that the page loads. The more interactive this information, the more likely that the client will wait for the full page loading.

Another way to optimize web system is to minimize code, both HTML, CSS and JS. To minimize the code, there are some services that allow you to remove all the spaces, paragraphs and formatting of the code text.

However, this kind of optimization is too time consuming and requires every time the system is upgraded and changed. The solution to this problem is to use a special WebPack tool with specific settings.

Compressing files will make them smaller in size and speed up loading. Combining some files will affect the number of files, which will also speed up the download, since fewer new requests and answers are required to download the next files.

To optimize the reload of each page with similar content change the application style to Single Page Application. We have two views - user and admin. View user – welcome.blade.php, admin – admin.blade.php. Depending on authorization, certain components will run on a particular page. We make changes to controller code and now they return welcome.blade.php or admin.blade.php depending on user rights. Next step is changing of links on those that are controlled by JavaScript. A program code below is showing usage of the fetch function during creating a React request and receiving a response from it:

```
this.state = {
/* Initialization State, which will include all products,
by default - empty array */
        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 });
        });
 }
```

When you create a query, response is provided, component changes the structure of DOM tree, and fetch function fills the pages with database data. Similarly, we are optimizing other pages and currently have a working system [15, 16].

The next optimization step is to create a balancing act between the cluster of servers and a client-side using the following algorithm:

— create a list of servers to connect;
— calculate the total number of possible servers;
— develop a method for random selection of the server;
— connect to the server;

— process the response from the server;
— in case of server failure write it in the list of non-working servers;
— make the same connection to another server;
— clear information about unavailable servers after a successful connection.

To do this, we implement a separate js file that contains an array listing the servers that are imported into the server selection program file with the following variables: brockenServers and randNum; the first is an empty array that contains non-working servers, the second is a random value. Next, we create a function that will determine a random number equal to the number of servers minus one (numbering in the array from zero). Next, check for a randomly selected number in the brokenServers array.

If the values are the same, the random number function is called [17]. It invokes the fetch function with the server address parameter that you want to connect to. The next step is to process the response from the server. If the server answered, the system continues to work, if not, the server index in the server array is written in the broken-Servers array and a new server is selected for connection. Server selection code for the connection:

```
let brokenServers = [];
let randNum = 0;
function getRandomNum() {
            randNum = Math.floor(Math.random()
            * servers.length);
}
getRandomNum ();
  if(brokenServers.length > 0) {
    for (let i=0; i <brokenServers.length; i++)  {
       if(brokenServers[i] === randNum) {
          getRandomNum();
       }
       else {
         getFetch(this);
         break;
       }
     }
   }
   else {
     getFetch(this);
   }
function getFetch(state) {
   fetch(servers[randNum] + 'api/')
   .then(response => {
      brokenServers = [];
      return response.json();
   }, function(reason) {
        brokenServers.push(randNum);
```

```
        return brokenServers;
   })
   .then(jsonResponse => {
     state.setState({categories: jsonRe-
sponse.categories});
     state.setState({products: jsonResponse.tovar});
   });
}
```

Test the work of balancing the server cluster on the client side. To do this, we specify two server addresses [14].

```
const servers = [
    "http://test-react-mag.loc/",
    "http://s1.test-react-mag.loc/"
];
```

The first address is the existing address where the system is hosted, the second is the missing non-response address. When connecting to the test-react-mag.loc server, the necessary data was received, otherwise, when connecting to the server located at s1.test-react-mag.loc, the server does not return an answer and its serial number is added to the non-working array servers. After successfully testing the load sharing between the servers on the client side, to complete this idea, you must deploy the system to multiple servers and specify their addresses instead of test-react-mag.loc and s1.test-react-mag.loc. This kind of balancing is characteristic only of the client side of the system. The averaged results of an experimental verification of the proposed system optimization method are presented in the table 1.

Table 1. The averaged results of an experimental verification of the proposed method

Parameters	**Before optimization**	**After optimization**
Load Time	3.606s	2.555s
First byte	0.539s	0.205s
Start render	1.900s	1.000s
First contentful paint	1.939s	0.997s
Speed index	3.030s	1.594s
Last painted hero	3.600s	1.866s
First CPU idle	> 1.939s	> 0.996s
Document complete		
Time	3.606s	2.555s
Requests	37	23
Bytes in	1.313 KB	1.017 KB
Fully loaded		
Time	3.853s	2.655s
Requests	38	24
Bytes in	1.313 KB	1.017 KB

As a result of implementation of the proposed web system optimization method we were able to obtain an improvement of the web system work of all considered criteria.

5 Conclusion

This paper describes methods for optimization of network web system which includes reduced load on server, developed sophisticated features that help improve user interface, save server traffic and funds. The scientific novelty of the paper lies in developed web system multicriteria optimization method which uses modern Front-End technologies. The practical value is optimized web system which uses a "smart" client, which can partially perform server logic, thereby providing fewer loads on the server and improve interactive properties of user interface. Optimization of system does not affect its features and consistency. The existing application architecture has been preserved, the conditions for the further improvement of system and its development have been created, and set tasks have been fully fulfilled.

References

1. Henderson, P.: Functional Programming Application and Implementation. Prentice Hall, 1980.
2. Tajima, M., Goto, K., Toyama, M.: Non-procedural generation of web pages with nested infinite-scrolls in superSQL. In: Proceedings of the 19th International Conference on Information Integration and Web-based Applications & Services, pp. 289-295 (2017). doi:10.1145/3151759.3151806
3. Larsen, H.L., Blanco, J.M., Pastor, R.P., Yager, R.R.: Using Open Data to Detect Organized Crime Threats: Factors Driving Future Crime. Springer. (2017)
4. Gorodnyaya, L.: On the Presentation of the Results of the Analysis of Programming Languages and Systems. In: CEUR Workshop Proceedings 2260, pp. 152-166 (2018)
5. Barki, H., Rivard, S., Talbot, J.: Toward an assessment of software development risk. Journal of management information systems 10(2), 203-225 (1993)
6. Zaninotto, F.: React is Slow, React is Fast: Optimizing React Apps in Practice. https://medium.com/dailyjs/react-is-slow-react-is-fast-optimizing-react-apps-in-practice-394176a11fba (2017) Accessed 07 Jan 2020
7. Ahmed, T.M., Bezemer, C.P., Chen, T.H., Hassan, A.E., Shang, W.: Studying the effectiveness of application performance management (APM) tools for detecting performance regressions for web applications: an experience report. In: IEEE/ACM 13th International Conference on Mining Software Repositories (MSR), pp. 1-12. IEEE (2016)
8. Elbaum, S., Karre, S., Rothermel, G.: Improving web application testing with user session data. In: 25th International Conference on Software Engineering IEEE Computer Society, pp. 49-59. IEEE (2003) doi:10.1109/ICSE.2003.1201187
9. Arora, A., Sinha, M.: Web application testing: A review on techniques, tools and state of art. International Journal of Scientific & Engineering Research 3(2), 1 (2012)
10. Atterer, R., Schmidt, A.: Tracking the interaction of users with AJAX applications for usability testing. In: Proceedings of the SIGCHI conference on Human factors in computing systems, 1347-1350 (2007)

11. Kirichek, G., Kurai, V.: Implementation quadtree method for comparison of images. In: 14th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET), pp. 129–132. IEEE (2018). doi:10.1109/TCSET.2018.8336171
12. Zhongsheng, Q.: Test case generation and optimization for user session-based web application testing. Journal of Computers 5(11), 1655-1662 (2010)
13. Kirichek, G., Tymoshenko, V., Rudkovskyi, O., Hrushko, S.: Decentralized System for Run Services. In: Luengo D., Subbotin S., Arras P., Bodyanskiy Ye., Henke K., Izonin I., Levashenko V., Lytvynenko V., Parkhomenko A., Pester A., Shakhovska N., Sharpanskykh A., Tabunshchyk G., Wolff C., Wuttke H.-D., Zaitseva E. (eds.) Proceedings of the Second International Workshop on Computer Modeling and Intelligent Systems (CMIS-2019), Zaporizhzhia, Ukraine, April 15-19, 2019, CEUR-WS.org 2353, pp. 860–872 (2019)
14. Kirichek, G., Harkusha, V., Timenko, A., Kulykovska, N.: System for detecting network anomalies using a hybrid of an uncontrolled and controlled neural network. In: Kiv, A.E., Semerikov, S.O., Soloviev, V.N., Striuk, A.M. (eds.) Proceedings of the 2nd Student Workshop on Computer Science & Software Engineering (CS&SE@SW 2019), Kryvyi Rih, Ukraine, November 29, 2019, CEUR-WS.org, pp. 138–148 (2019)
15. Qian, Z., Miao, H., Zeng, H.: A practical web testing model for web application testing. In: Third International IEEE Conference on Signal-Image Technologies and Internet-Based System, pp. 434-441. IEEE (2007). doi:10.1109/SITIS.2007.16
16. Polpong, J., Kansomkeat, S.: Syntax-based test case generation for web application. In: 2015 International Conference on Computer Communications and Control Technology (I4CT), pp. 389-393. IEEE (2015). doi: 10.1109/I4CT.2015.7219604
17. Kulykovska N.A, Timenko A.V. A Structure of Semantic Service in a Distributed Knowledge Based System. In: Luengo D., Subbotin S., Arras P., Bodyanskiy Ye., Henke K., Izonin I., Levashenko V., Lytvynenko V., Parkhomenko A., Pester A., Shakhovska N., Sharpanskykh A., Tabunshchyk G., Wolff C., Wuttke H.-D., Zaitseva E. (eds.) Proceedings of the Second International Workshop on Computer Modeling and Intelligent Systems (CMIS-2019), Zaporizhzhia, Ukraine, April 15-19, 2019, CEUR-WS.org 2353, pp. 533-544 (2019)