International Journal of Engineering and Management Research e-ISSN: 2250-0758 | p-ISSN: 2394-6962
Volume-12, Issue-6, (December 2022)
www.ijemr.net https://doi.org/10.31033/ijemr.12.6.21

# Virtual DOM Implementation in KissJS

M.W.M.R.C.T.B. Madigasekara[1], S.P.P.P. Wanigarathne[2], P.D.G.N.T.D. Dharmasinghe[3], Dr. Nuwan Kodagoda[4], A.G.S.D. Wickramarathna[5] and Dr. Jeewaka Perera[6]
[1]Faculty of Computing, Department of Software Engineering, Sri Lanka Institute of Information Technology, SRI LANKA
[2]Faculty of Computing, Department of Software Engineering, Sri Lanka Institute of Information Technology, SRI LANKA
[3]Faculty of Computing, Department of Software Engineering, Sri Lanka Institute of Information Technology, SRI LANKA
[4]Faculty of Computing, Department of Software Engineering, Sri Lanka Institute of Information Technology, SRI LANKA
[5]Faculty of Computing, Department of Software Engineering, Sri Lanka Institute of Information Technology, SRI LANKA
[6]Faculty of Computing, Department of Software Engineering, Sri Lanka Institute of Information Technology, SRI LANKA

[1]Corresponding Author: chandula.madigasekara@gmail.com

**ABSTRACT**

**In this fast-paced modern world, time is a major asset for most people, including web developers. When developing web applications, the language or framework being used should help the developer create applications fast and efficiently, but at the same time it should be simple enough for them to grasp it quickly.**

**This proposed framework takes into consideration the pros and cons of modern Single Page Application (SPA) development frameworks and tries to use some their approaches to achieve efficiency, and at the same time use its own ways to simplify other tasks that are complex in the aforementioned frameworks.**

**The main objective this framework will try to accomplish is simplify the state management process. However, this paper will focus on the change detection aspect of the framework. The method of change detection used in the framework is the virtual DOM. Reasons for moving forward with this approach will be discussed later in the paper.**

***Keywords*—** SPA, Virtual DOM, Change Detection

## I. INTRODUCTION

Since the creation of JavaScript back in 1995, it has greatly evolved into one of the most used programming languages worldwide [1]. As it has been more than 20 years since its making, a great number of libraries and frameworks have been implemented using JavaScript as the language [1]. A subset of these frameworks are frontend UI frameworks. Most modern websites leverage JavaScript in order to make webpages more interactive and to have a wide range of interactivity [2].

Traditional websites are usually a collection of webpages, where a new HTML document is loaded every time its content changes [2], however, this approach is very resource intensive. A majority of the modern web applications however, use a Single Page Application (SPA) development model [2]. This approach revolves around using a single HTML document where only components that are changed are updated, thus, drastically reducing the application loading speed [2]. Single page applications are increasing in popularity in current times, and the JavaScript frontend frameworks they are built from are also becoming increasingly popular. Many people would agree that from these frameworks, the top three most popular frameworks are React, Vue and Angular [7, 8].

With the emergence of all these new frameworks SPAs are becoming more prevalent, also due to their various advantages such as dynamic loading, URL routing, HTML rendering and the caching of data for faster loading times [9]. And the three frameworks mentioned above are the most popular frameworks to use for the development of SPAs so they will be the main comparison for this research.

This research component will focus on effectively re-rendering components, that is using the most efficient approach to re-render components, minimizing the time taken for changes to be seen. The following sections of this paper will look at the different approaches available for change detection and selecting the most suitable approach.

## II. RELATED WORK

When taking into consideration the re-rendering of DOM elements, the framework must first determine if re-rendering of that element is necessary. According to the article published by Tero Parviainen [5], a multitude of change detection mechanisms are available for use in JavaScript frameworks. Some of these methods include: Server-side rendering, manual re-rendering, data binding, dirty checking, virtual DOM and key-value observation (KVO).

Out of the above-mentioned methods, only dirty checking, virtual DOM and KVO were considered for the change detection mechanism, since these are the most prominent and commonly used techniques.

- ***Key-Value Observation (KVO)***

This mechanism utilizes change events and change listeners to make the necessary changes once updates have been detected. This uses the Publisher/Subscriber design pattern, and by doing so it maintains an event channel between the objects that fire events and the objects that want to receive those notifications [3].

Using this method, whenever some action in the UI changes some underlying data, a change event is triggered and the model is updated, thus updating all the views that are displaying the specific data [3].

- ***Dirty Checking***

This method is more commonly used by the Angular framework. With this, whenever there is a possibility that data has changed, a loop known as the change or digest cycle is executed to see if any of the data has actually changed. This process, however is quite expensive since it has to traverse all the data that is being watched and compare them to their previous values. So, it is crucial that this cycle is executed only when needed [3].

The actual "dirty checking" will happen inside the digest cycle. Once it has detected that the current value of some data differs from its previous value, its watcher is notified once the digest cycle is complete. The advantage of notifying watchers after a cycle is completed is that multiple changes can be detected in one cycle [3].

The process following after a watcher has been notified is it would update the model, and this in turn would fire off other listeners. This dirty checking approach gives Angular its two-way data binding capability. This means that whenever a watched element is changed in the DOM, the change is reflected in the model, and the vice-versa holds true.

- ***Virtual DOM***

The concept of the virtual DOM is not something that is new, however it was only made relevant by Facebook, now known as Meta, with their creation of React [6]. With this method a vanilla JavaScript object is constructed that is a representation of the DOM, and this is known as the virtual DOM. Each time a change occurs a new virtual DOM is constructed and the new and previous copies and compared in order to identify the changes. Once they have been identified they are used to update the actual DOM.

The virtual DOM is only part of the re-rendering process. In order to fully utilize the virtual DOM a diff algorithm must be introduced to effectively compare the two virtual DOMs so that the differences can be found as efficiently as possible.

Various researches have been carried out in order to find which change detection mechanism was indeed the best [3, 4]. From the results of these papers we can conclude that one method cannot be selected as the best, as each approach excels at certain scenarios but does not perform as well in others. However, by considering the overall performances of the mechanisms in all the scenarios tested, it could be concluded that the virtual DOM is a valid candidate to ensure minimal re-render times. For that reason, the change detection mechanism used for this framework was the virtual DOM.

## III. METHODOLOGY

As mentioned previously, in order to proceed with change detection, the framework uses the virtual DOM approach as opposed to the other methods available due to its better overall performance. However, the virtual DOM by itself cannot handle the change detection process, it is only half of the full mechanism.

To fully utilize the virtual DOM and maximize its efficiency, a diff algorithm was implemented. The purpose of this algorithm is to find the changes that were made to the elements in order to reflect those changes in the actual DOM.

In order to work in harmony with the diff algorithm, the virtual DOM was implemented as follows, the virtual DOM was a collection of virtual DOM nodes that were nested in each other representing a tree-like structure, similar to that of the original DOM. Each virtual DOM node represents an actual node in the DOM. Each virtual DOM node has the following attributes: a type, tag name, attributes and children. There are only 2 types of nodes in this framework, Node type and Text type. Node type nodes will have other nodes nested within them, whereas Text type nodes will only have a text value, meaning they are the final or leaf nodes of the virtual DOM tree. The tag name attribute basically identifies what kind of HTML element it is, for example, a div (<div>) or a list item (<li>). The children attribute is an array of nodes that are nested directly under the current node.

So as to actually detect changes, two copies of the virtual DOM are made. The latest changes to the nodes are stored in one copy while the previous changes are stored in the other. The diff algorithm then loops through both copies in order to identify which nodes have undergone any changes.

In the framework, for a change to be detected, it must be in one of the following four forms: a node was appended, replaced, or removed from the virtual DOM, or there was a change in the value of the node's attributes or text. Once one of these changes is detected the framework will add these changes to an array of Patches. Each Patch contains information on the node that has changed and what kind of change has occurred.

As soon as the diffing process is over the array of Patches is looped through, and the respective nodes are updated depending on the kind of change that has occurred. When these updates are done the original DOM will be updated and this completes one successful cycle of the change detection mechanism implemented in this framework.

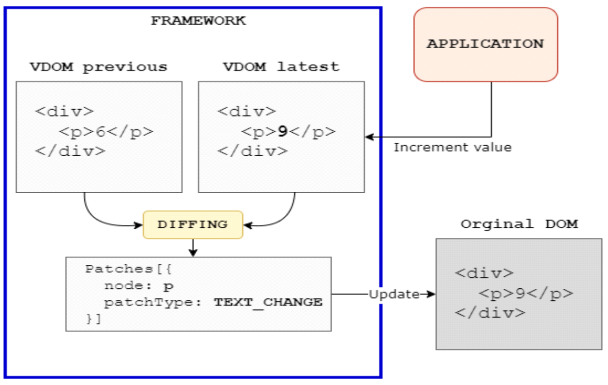

**Figure 1:** High level overview of change detection mechanism

As mentioned previously the diffing process will be handled with the help of a diffing algorithm. It handles the diffing by comparing two nodes of the two virtual DOMs that are at the same hierarchy level in the virtual DOM.

Firstly, a conditional statement checks if the two nodes are exactly equal to one another, meaning both their types and values are the same. Since the framework implementation language used was JavaScript, the "===" operator was used for this check. If the nodes are indeed equal no further operations will be performed and the algorithm will move onto the next pair of nodes.

In the case that the check fails, it indicates that the two nodes are different, meaning a change will have to be made. When this happens, the first condition hecked is the type of the node, i.e. if it is a Node type a certain set of actions will take place, if it is a Text type a different set of actions will occur.

If the node in question is of type Text, the procedure is simple. A Text node with the updated value is appended to the array of Patches. If the node is of type Node, a check is first performed to see if there is a change in attributes. In the case that it is so, it will be added to the array of Patches and if not no changes will occur. Next, the children of the node are checked. The algorithm runs recursively so the above steps are repeated for the children of the node.

## IV. RESULTS AND DISCUSSION

For the purpose of measuring the performance of the framework's change detection mechanism, it was tested against the frameworks React and Vue, in order to gain an idea about where the framework stands in terms of the efficiency of rendering elements.

A simple application was created using the different frameworks. It consisted of four buttons: "Add 5", "Add 10", "Add 100", "Reset". On the click of these buttons the application would render five, ten, one hundred and remove Todo items, respectively. Each Todo item was a simple div with a border and a text indicating which Todo it was (for example: Todo 5).

The tests were run on an HP Pavilion 15ccx with the following specifications:

- Processor – Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80Hz
- Installed memory(RAM) – 8.00 GB
- System type – 64-bit Operating System, x64-based processor

The render times were measured using the Performance tools of the Google chrome devtools. This gave the capability to record actions on the screen and would give various measurements including the render times.

The method in which the test was carried out was the application was served on the localhost and the "Add 5" button was clicked, afterwards the "Reset" button was clicked. The render time for this operation was then recorded. These steps were then repeated without restarting the application. The purpose of this is to get the render times with the initial load and after the initial load.

The steps mentioned above were repeated five times for each button in order to get an average estimate for the render times. The results obtained are as follows.

**Table I:** Average render times for react application

| **Action** | **Average Render Time (ms)** | |
|---|---|---|
| | ***Initial Load*** | ***Normal*** |
| Add 5 | 20.2 | 8.0 |
| Add 10 | 22.4 | 8.6 |
| Add 100 | 23.0 | 13.2 |

**Table II:** Average Render times for Vue application

| Action | Average Render Time (ms) | |
|---|---|---|
| | ***Initial Load*** | ***Normal*** |
| Add 5 | 11.4 | 7.5 |
| Add 10 | 12.0 | 8.2 |
| Add 100 | 16.0 | 11.6 |

**Table III:** Average render times for kissjs application

| Action | Average Render Time (ms) | |
|---|---|---|
| | ***Initial Load*** | ***Normal*** |
| Add 5 | 12.6 | 7.6 |
| Add 10 | 13.4 | 8.4 |
| Add 100 | 14.8 | 9.0 |

From the results obtained, it can easily be seen that the React application clearly takes the most time to render the items on the initial load. In comparison to that, the Vue application rendered items much faster on the initial load. However, when rendering the one hundred Todos, the KissJS application was recorded having the lowest render time.

Similar results can be observed for the normal render times. React having the highest times, although the difference is minute compared with the times observed for the other two frameworks. The Vue application recorded the least times again, except for the one hundred Todos, where the KissJS application recorded the least time.

When taking into consideration all the results obtained overall, the difference in the render times is insignificant. However, the difference in the initial load times was quite noticeable, with the React application taking approximately 9 milliseconds longer than the other two applications in all the actions performed.

The KissJS application fared quite well in comparison to the Vue and React application, with it having faster render times than the React application in all the tests and just slightly higher times than the Vue application. Nonetheless, it managed to maintain the lowest render times for when the one hundred Todos were generated.

React provides an in-built solution to allow the developer to control which elements they want to re-render when a change occurs, the shouldComponentUpdate() method. This function was not implemented in the React application that was tested since the application was quite small and it is used to maximize the performance of a large application. The implementation of the function in this application would have only lowered the normal render times and not the initial load times.

The performance of all the applications could also have been improved by using a PC with better specifications.

## V. CONCLUSION

In conclusion, the change detection mechanism of the newly created framework, KissJS, is the virtual DOM. This was chosen due to its better overall performance in comparison to the other change detection methods available, such as KVO and dirty checking.

The virtual DOM along with the diff algorithm implemented for the framework provides the framework with an efficient means to detect changes in the underlying code and reflect them in the DOM.

The performance of the KissJS virtual DOM and diff algorithm was tested by creating the same application with the KissJS, React and Vue frameworks and comparing the render times of performing different actions. React and Vue were chosen as the frameworks to compare with since they also implement a virtual DOM and diff algorithm.

From the results obtained it could be seen that the KissJS application recorded faster times than the React application and only slightly slower times than the Vue application. A key observation was that the KissJS application recorded the fastest times when more elements needed to be rendered on the screen. It could be deduced from this that the framework scales well when the application grows. This was another goal of the framework's change detection mechanism, i.e. to maintain render times as the application grows. This was also another reason that the virtual DOM was selected for this framework since frameworks that utilize other change detection strategies, such as Angular with dirty checking, tend to take longer to find and render changes as the application gets larger.

The KissJS framework, therefore has a fairly efficient change detection mechanism, and with future work and optimizations the render times could be further improved.

## ACKNOWLEDGMENT

This research was supported by the Sri Lanka Institute of Information Technology CDAP module. I am especially thankful to our research supervisor, Dr. Nuwan Kodagoda, and co-supervisor, Dr. Jeewaka Perera, for the continued support they provided throughout the duration of the project. I would also like to thank my team members for the support they provided me in completing this component of the framework.

## REFERENCES

[1] E. Wohlgethan. (2018). Supporting web development decisions by comparing three major javascript frameworks: angular, react and Vue.js. *Hamburg University of Applied Sciences, J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892,* pp.68–73.
[2] E. Saks. (2019). JavaScript frameworks: Angular vs React vs Vue. *University of Applied Sciences*.
[3] M. Grov. (2022). Building user interfaces using virtual DOM. *M.S. Thesis, Dept. of Inform., Univ. Oslo*. Accessed on Jan. 20, 2022. [Online]. Available: https://www.duo.uio.no/handle/10852/45209.
[4] D. Muyldermans.(2019). How does the virtual DOM compare to other DOM update mechanisms in JavaScript frameworks?. Accessed: Jan. 24, 2022. [Online]. Available: http://www.daisyms.com/THESIS.pdf.
[5] “Change And Its Detection In JavaScript Frameworks,” teropa.info. Available at: https://teropa.info/blog/2015/03/02/change-and-its-detection-in-javascript-frameworks (accessed Jan. 22, 2022).
[6] R. Hakulinen. (2022). Using immutable data structures to optimize angular change detection. *M.S. Thesis, Tampere Univ. Tech*. Accessed on Jan. 19, 2022. [Online]. Available: https://trepo.tuni.fi/bitstream/handle/123456789/26217/Hakulinen.pdf?sequence=4.
[7] “Top 10 Most Popular JavaScript Frameworks for Web Development - GeeksforGeeks.” https://www.geeksforgeeks.org/top-10-most-popular-javascript-frameworks-for-web-development/ (accessed Jan. 19, 2022).
[8] “Best Front End Frameworks For Web Development - InterviewBit.” https://www.interviewbit.com/blog/best-front-end-frameworks/ (accessed Jan. 19, 2022).
[9] “Top 5 Single Page Application Frameworks To Use in 2022.” https://www.monocubed.com/top-single-page-application-frameworks/ (accessed Jan. 22, 2022).