Journal of Computer Science Engineering and Software Testing
e-ISSN: 2581-6969
Volume-6, Issue-3 (September-December, 2020)

www.matjournals.com
https://doi.org/10.46610/JOCSES.2020.v06i03.001

Performance Comparison Between Development Approaches in React: Hooks, Functional and Classes

Jason Aaron Ancona Reyes
PG Student, Department of Computer Engineering, Universidad Da Vinci, Mexico City, Mexico

*Corresponding Author: jasonaar@hotmail.com

ABSTRACT

Nowadays, due to constant innovations in technology, there are many available libraries to help developers in the process of designing and code visual and functional websites. Among all of them, React is currently one of the most popular in the developer community. When developers work with React, there are three common approaches used in the community. The purpose of this project is to provide a comparison between those approaches and an insight of how they perform in a real world situation. For this task, chrome developer tools was used for debugging. The three different approaches (Classes, functional and Hooks components) were tested using 4 different projects. Each project had 3 different versions. One solely relies on state management, the second one on an API response and the third one an API call and the use of Redux. After performing all tests, Hooks was the clear winner overall but still there are developers which use the other two approaches, classes approach performed better than functional but it can lead to misuse of class lifecycle which can result in a performance downgrade, that is why if possible, the use of functional when the manage of state is not required is still recommended instead of class based components for that work. React is being updated regularly and further improvement may be expected. Further studies may be needed to cover new incoming features, optimizations and improvements.

Keywords-- Hooks, javascript, programming, react, web

INTRODUCTION

Web development has been constantly evolving over the span of years, however, with this evolution, many tools emerged to facilitate web development and make it attractive to users, JavaScript is the most commonly encountered client-side computer language which is used by the majority of web sites and it is supported by all modern browsers with this increase in use and popularity, it was inevitable that a large amount of tools to help in the development were published [1, 2]. They can be divided into different categories: IDEs and editors, package managers, compilers, bundlers, libraries and frameworks. Among such tools, React is the one whose popularity has risen and is currently known and used worldwide. In addition to the previous, it is maintained by Facebook and a community of individual developers and companies. React is a JavaScript library which purpose is to improve the process of developing reusable user interface (UI) components, and, according to React official site, provides a declarative API so that one do not have to worry about exactly what changes on every update. This makes writing applications a lot easier, it is currently one of the most popular Javascript libraries available and competes with such as Angular and Vue Js, that is why in this work is focused on the analysis of React, this library since it is release date has been constantly updated and supported by it is team, while originally there was just one standard approach of developing, over the updates, new approaches of developing components were introduced, and all of them are still being supported, that is why this work is focused on testing each approach to evaluate and deliver an insight of how each approach performs and compare them to each other [3].

APPROACHES OF DEVELOPMENT IN REACT

The first version of react to support hooks is 16.8, “Hooks” is an extension of the functional components approach released in the version 0.14 of React [4]. Functional components allow developers to create components that contains methods and events without declaring a class. The main disadvantage of doing this was the inability to handle intern changes of state except by depending on a parent component which provides properties to the child functional component. With hooks, it is possible to provide functional components a way of handle states and methods which were not available before hooks [5]. Previously to these approaches there was just one way of declaring components, which was “Classes”. The main advantage of classes is the ability to manage its own internal state, but it comes with the disadvantage that every component

https://doi.org/10.46610/JOCSES.2020.v06i03.001

created this way, is not completely stateless, which if misused can return serious performance problems in the whole project [6]. When the previous statement is considered, that is why there are three actual used approaches of developing in react:

Hooks

Hooks is a reinvention of functional components, the implementation of hooks let functional components to be able of handling it is own state and exhibit features which used to be exclusive of class based components [4].

Functional and Classes Components

While hooks approach solely relies on functional components, this one relies on class components when management of state or a class lifecycle method is needed and functional components when neither of the before mentioned is required, the component just use internal or external methods and data provided from parent or other source [5].

Classes Components

This approach relies solely on class components, even when it is not necessary the ability to handle own state, the created component will always be able to do it but it is a decision from the developer to use this capabilities or not, whole projects made with this approach will be made of class based components [6].

METHODOLOGY OF PERFORMANCE COMPARISON

In order to compare the development approaches in React, the three previously mentioned approaches were carried out, but to correctly measure the result, four different projects were used to measure the effectiveness of each one of them, all projects have Ant design as a library of components of UI for the realization of interfaces.

- In the first project all the components are based on React classes, however, it uses Antd version 3.6.7 being that after version 4 several of its components were refactored to Hooks.
- In the second project, the components are identical to those of the project, however, it has version 4.6.1 of Antd being the version at the time this paper was done.
- In the third project, the components that require state management were made using class-based components and those without the need of internal state using functional components.
- The fourth project is done entirely in Hooks.

For all projects, 3 different versions were made, which are:

- First one only consists of a page which has two counters.

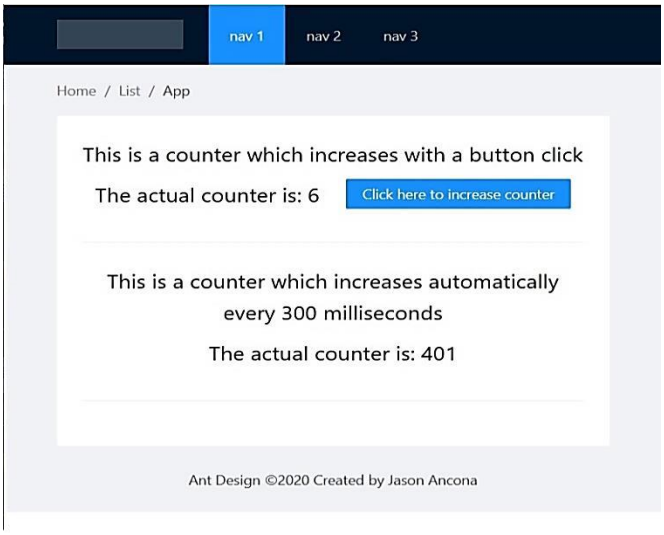

Figure 1: *First version of the project.*

As seen in the Fig. 1, the first counter is updated +1 when an event (click) is made on the blue button, and the other counter is updated +1 every 300 milliseconds. The Second one, uses the external libraries React-router-dom and Axios, in addition to include what was done in the previous version, it consists of a new page using Axios as an HTTP client, a request is sent to a local server that returns an array of 5 posts, those posts are added into the current state and displayed in the page.

https://doi.org/10.46610/JOCSES.2020.v06i03.001

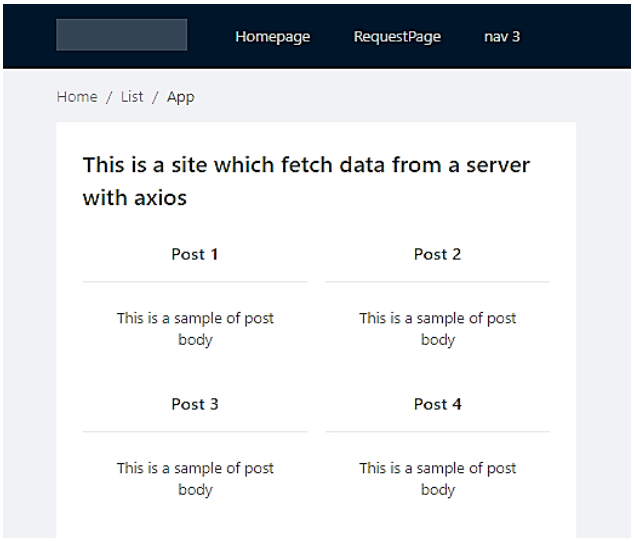

Figure 2: *Second version of the project.*

The Fig. 2, shows how the page looks when the request has been solved successfully and the posts are displayed on the page. The third, in addition to including the two previous versions, now three external libraries are added and used which are: Redux, React-redux and Redux-saga, one more page is created whose function is to execute a Redux action that triggers a Redux Saga action which makes a request to a local server. Similar to the previous project, the request returns an array of 5 posts and those are displayed on the page, but instead of storing them in the local state, they are stored in redux.

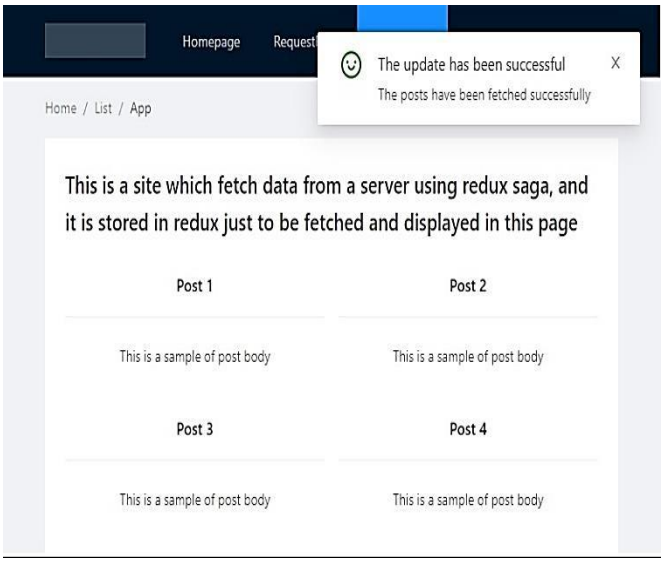

Figure 3: *Third version of the project.*

In Fig. 3, it is shown how the page looks when the request has been solved successfully, taking advantage of the capabilities of Redux. It displays a notification if the array of posts is stored in Redux successfully.

- The hardware used for testing is: Processor: AMD R7 3700x
- RAM: 16gb RAM 3000mhz
- Storage: 1TB ssd HP ex820
- Motherboard: Asus PRIME X570-P
- OS:Windows 10 Professional edition

Testing Environment

Tests were made in browser Google Chrome version 85.0.4183.121 (64 bits) thanks to its developer's tools it is possible to reload a page and do a profiling of the page until it is loaded or until the user interrupts the operation, the heap snapshot is generated after a profiling is done for the actual

https://doi.org/10.46610/JOCSES.2020.v06i03.001

test. In order to provide a situation which is near to a real-life usage, each test was carried out 30 minutes after the previous one. If tests were done in a shorter time span, results can be compromised to be influenced by cache memory or memory actual allocation, each version of each project was tested 10 times and from all tests the average of all parameters is obtained and displayed in tables. For testing the first version in all projects, total profiled time was managed entirely by the browser, for the other versions it was a set period of time considering that those versions are dependant of some external responses and third package libraries added to projects.

RESULTS AND COMPARISON

The performance of the test in the first version had these results:

Table 1: *First project testing results.*

Parameter	Classes	Classes updated	Classes + ES6	Hooks
Loading (ms)	16	14.2	18.9	14.1
Scripting (ms)	669.4	596.9	568.8	490.5
Rendering (ms)	17.5	20.9	17.5	17
Painting (ms)	2.4	3.7	3	2.4
System (ms)	74.2	64.7	121.1	84.6
Idle (ms)	320.5	289.8	269.9	200.3
total (ms)	1100	990.2	999.2	808.9
Total without idle (ms)	779.5	700.4	729.3	608.6
Heap snapshot Size (MB)	9.84	8.09	8.06	8.02
Total build size (KB)	319.688	159.638	159.616	159.456

From Table 1, it is possible to infer that the approach which has the better performance in time, memory usage in snapshot, and total project size is the hooks approach. While classes using an outdated Antd had the poorest performance even doubling the project size, using an updated version had better results almost performing better than the project with ES6 syntax and almost matching it is project size. When performing tests in second version, it was opted to keep an average profiled time of 1500 milliseconds in each test to avoid excluding the time of re-rendering after obtaining the information from a local API. The performance of the test in the second version had these results:

Table 2: *Second project testing results.*

Parameter	Classes	Classes updated	Classes + ES6	Hooks
Loading (ms)	17.1	16.7	17.8	19.7
Scripting (ms)	757	604.7	619.1	553.2
Rendering (ms)	18.9	22.2	17.5	18.7
Painting (ms)	2.8	3.6	3.7	3.3
System (ms)	79.7	92.4	103.4	86.8
Idle (ms)	627.1	765.3	741.4	821.6
total (ms)	1502.6	1504.9	1502.9	1503.3
Total without idle (ms)	875.5	739.6	761.5	681.7
Heap snapshot Size (mb)	9.38	7.37	7.43	7.42
Total build size (KB)	340.398	181.218	181.456	180.986

From Table 2, it is possible to notice that hooks approach still has the best performance in time, memory usage, and total project size. Classes approach still obtains the worst performance in all subjects, there is a change in the updated version, now it outperform in all subjects the project using ES6 syntax and even outperform in heap snapshot size the hooks version, meanwhile the results still the same as the first test, hooks still has the advantage, succeeded by Class updated, then ES6 and lastly classes. When performing tests in the third version, it was opted to keep an average profiled time of 3000 milliseconds in each test to avoid excluding the time of re-rendering and the manage of internal project state done by redux.

https://doi.org/10.46610/JOCSES.2020.v06i03.001

The performance of the test in the third version had these results:

Table 3: *Third project testing results.*

Parameter	Classes	Classes updated	Classes + ES6	Hooks
Loading (ms)	17	21.1	16.3	16.4
Scripting (ms)	880.9	877.7	900.7	810.8
Rendering (ms)	40.3	36.1	36.8	35.5
Painting (ms)	6.4	7.7	6.7	6.7
System (ms)	160	105.4	113.4	122.9
Idle (ms)	1900.3	1959.5	1853.3	2014.8
total (ms)	3004.9	3007.5	2927.2	3007.1
Total without idle (ms)	1104.6	1048	1073.9	992.3
Heap snapshot Size (mb)	9.41	10.18	10.5	10.5
Total build size (KB)	355.797	203.427	202.316	201.726

From Table 3, it is possible to notice that hooks outperforms all other approaches similarly as done in the first version. Curiously in this version Classes obtain the less size in heap snapshot, but still gets the worst performance in overall test, this could be thanks to Antd 3.6.7, which does not have Antd icons integrated, and thanks to this, the usage of an icon were removed from that project, still, the updated class version still obtain less snapshot size than hooks and functional approach, but in size and milliseconds of usage still hooks the clear winner.

CONCLUSION

The purpose of this paper is to provide an insight of how each approach of programming in react impact in the release build of a project. The three most common approaches of developing in React were evaluated and anti-pattern approaches were omitted, such as using class components for stateless components and hooks for state managements were not tested because they are not part of the React developer team good practices. Hooks is clearly the winner in overall tests, this could be inferred thanks to hooks being the most recent approach, but still other approaches still benefits from updates so keeping the library and components up to date may be beneficial for old and new projects. Contrary to expected, classes approach had better results than functional, but React team advise the use of functional and hooks components over classes. This recommendation is made to avoid the use of life cycle methods in class components which are difficult to use correctly and the misuse of them can lead to performance issues as seen in the old version of Antd. Reviewing different approaches has given us different results, which prove that React has been improving over the time such as hooks has been the last major update at the time this paper was done.

DISCUSSION AND FUTURE WORK

As mentioned, React is still evolving together with web and javascript and currently is one of the most popular libraries available and it is community keep growing over time, further tests may be required when a new feature or new improvement has been done to the library, while currently class components are still supported and developers team mention that they plan to keep this support for future versions, they advice to use hooks for new works and use classes for legacy code support exclusively, in the future this feature may be compromised due to next features are destined to improve hooks or even may bring new development approaches in React, this may serve as a motivation to new and old users to keep up to date with new features the library may bring in next updates.

REFERENCES

1. Ku. Chhaya A. Khanzode, Ravindra D. Sarode (2016), “Evolution of the world wideweb: From web 1.0 to 6.0”, *Internat. J. of Dig. Lib. Ser.*, Volume 6, pp. 1-11.
2. D. Mitropoulos, P. Louridas, et al. (2019), “Time present and time past: Analyzing theevolution of javascript code in the wild”, *IEEE/ACM 16th International Conference on Mining Software Repositories*.
3. Facebook, “React Github” Facebook, [Online] Available from: https://github.com/facebook/react/.
4. Facebook, “Hooks FAQ” Facebook, [Online] Available from: https://en.reactjs.org/docs/hooks-faq.html.
5. Facebook, “React v0.14”, Facebook, [Online] Available from: https://es.reactjs.org/blog/2015/10/07/react-v0.14.html.
6. Facebook, “React Component” Facebook, [Online] Available from: https://es.reactjs.org/docs/react-component.html.