A SEMINAR REPORT ON

Performance Optimization Techniques for ReactJS

Submitted By

VISHNU VISWAMBHARAN

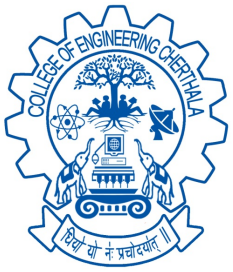
Reg. No . CEC17CS065

under the esteemed guidance of

Mr. MUHAMMED ILYAS H

Assistant Professor

Department of Computer Engineering



November 2020

DEPARTMENT OF COMPUTER ENGINEERING
COLLEGE OF ENGINEERING,CHERTHALA
PALLIPPURAM P O, ALAPPUZHA-688541,
PHONE: 0478 2553416, FAX: 0478 2552714
http://www.cectl.ac.in

A SEMINAR REPORT ON

Performance Optimization Techniques for ReactJS

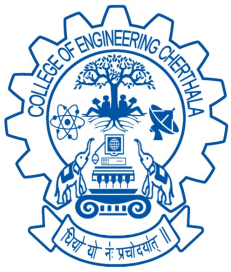
Submitted By

VISHNU VISWAMBHARAN *(Reg. No. CEC17CS065)*

under the esteemed guidance of

Mr. MUHAMMED ILYAS H

In partial fulfillment of the requirements for the award of the degree
of
Bachelor of Technology
in
Computer Science and Engineering
of
APJ Abdul Kalam Technological University

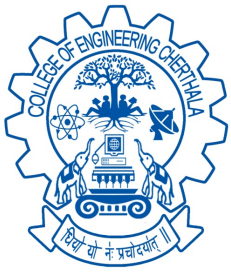


November 2020
Department of Computer Engineering
College of Engineering, Pallippuram P O, Cherthala,
Alappuzha Pin: 688541,
Phone: 0478 2553416, Fax: 0478 2552714
http://www.cectl.ac.in

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

COLLEGE OF ENGINEERING CHERTHALA

ALAPPUZHA-688541



C E R T I F I C A T E

This is to certify that, the seminar report titled **Performance Optimization Techniques for ReactJS** is a bonafide record of the **CS405 SEMINAR & PROJECT PRELIMINARY** presented by **VISHNU VISWAMBHARAN** (Reg.No.CEC17CS065), Seventh Semester B. Tech. Computer Science & Engineering student, under our guidance and supervision, in partial fulfillment of the requirements for the award of the degree, **B. Tech. Computer Science & Engineering** of **APJ Abdul Kalam Technological University**.

Guide	**Co-ordinator**	**HoD**
Mr. Muhammed Ilyas H	**Greeshma N.Gopal**	**Dr. Priya S**
Assistant Professor	Assistant Professor	Professor
Dept of Computer Engg.	Dept of Computer Engg.	Dept of Computer Engg.

ACKNOWLEDGEMENT

This work would not have been possible without the support of many people. First and the foremost, I give thanks to Almighty God who gave me the inner strength, resource and ability to complete my seminar successfully.

I would like to thank **Dr. Mini M G**, The Principal, who has provided with the best facilities and atmosphere for the seminar completion and presentation. I would also like to thank HoD **Dr. Priya S** (Professor, Department of Computer Engineering) and my seminar guide **Mr. Muhammed Ilyas H** (Assistant Professor, Department of Computer Engineering), my seminar coordinator **Mrs.Greeshma N.Gopal** (Assistant Professor, Department of Computer Engineering) for the help extended and also for the encouragement and support given to me while doing the seminar.

I would like to thank my dear friends for extending their cooperation and encouragement throughout the seminar work, without which I would never have completed the seminar this well. Thank you all for your love and also for being very understanding.

ABSTRACT

React is one of the popular web frameworks that has gained importance over other frameworks such as Angular, Vue, etc.. This is because of its implementation of Virtual DOM, whose primary objective is to enhance the overall performance of the application. However, there are certain things that one has to keep in mind before designing the applications. Failing to anticipate the problems that may occur component hierarchy will lead to performance degradation. Some of the commonly faced problems are component re-rendering, application lag due to background computations being run, lag due to processing large data sets in a single stretch, etc.

This paper will describe some of the practical ways of overcoming such problems within the application, thus enhancing the performance of the ReactJS App in a production environment.

Keywords– ***React.js,optimization, javascript, web application***

Contents

List of Figures

Chapter 1

INTRODUCTION

ReactJS is a declarative, efficient, and flexible JavaScript library for building reusable UI components. It is an open-source, component-based front end library responsible only for the view layer of the application. It was created by Jordan Walke, who was a software engineer at Facebook. It was initially developed and maintained by Facebook and was later used in its products like WhatsApp Instagram. Facebook developed ReactJS in 2011 in its newsfeed section, but it was released to the public in the month of May 2013.

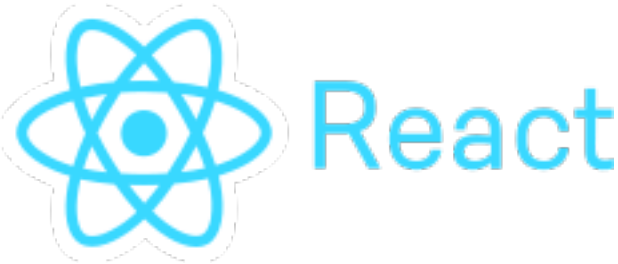


Fig. 1.1: React logo [1]

Today, most of the websites are built using MVC (model view controller) architecture. In MVC architecture, React is the 'V' which stands for view, whereas the architecture is provided by the Redux or Flux.

A ReactJS application is made up of multiple components, each component responsible for outputting a small, reusable piece of HTML code. The components are the heart of all React applications. These Components can be nested with other components to allow complex applications to be built of simple building blocks. ReactJS uses virtual DOM based mechanism

to fill data in HTML DOM. The virtual DOM works fast as it only changes individual DOM elements instead of reloading complete DOM every time.

Internally, React uses several clever techniques to minimize the number of costly DOM operations required to update the UI. For many applications, using React will lead to a fast user interface without doing much work to specifically optimize for performance. But while building complex applications, there are many ways to significantly improve the performance of application. Some of these techniques can only be applied to class based components while others are general techniques.This paper will cover some of the most important optimization techniques that should be followed while building a React application

Chapter 2

BASICS OF REACT

React is a popular javascript library that was mainly designed to address the performance issues in the web application.The main advantage of React is the splitting of code into resuable components and the speed rendering pages. React uses virtual DOM that decides whether the component has to be reloaded or not based on the current state of the component and the changes that have occurred. This prevents the application from re-rendering unnecessarily. Apart from this React also introduces one way data flow which helps to control the flow of the data within the application which makes the tracking of the occurred easier and also simplifies the propagation and the stability.

2.1 React Components

Earlier, the developers write more than thousands of lines of code for developing a single page application. These applications follow the traditional DOM structure, and making changes in them was a very challenging task. If any mistake found, it manually searches the entire application and update accordingly. The component-based approach was introduced to overcome an issue. In this approach, the entire application is divided into a small logical group of code, which is known as components.

A Component is considered as the core building blocks of a React application. It makes the task of building UIs much easier. Each component exists in the same space, but they work independently from one another and merge all in a parent component, which will be the final

UI of your application.
In ReactJS, we have mainly two types of components.

2.1.1 Functional Components

In React, function components are a way to write components that only contain a render method and don't have their own state. They are simply JavaScript functions that may or may not receive data as parameters. We can create a function that takes props(properties) as input and returns what should be rendered.

2.1.2 Class components

Class components are more complex than functional components. It requires you to extend from React. Component and create a render function which returns a React element. You can pass data from one class to other class components. You can create a class by defining a class that extends Component and has a render function.

2.2 React Props

Props stand for "Properties." They are read-only components. It is an object which stores the value of attributes of a tag and work similar to the HTML attributes. It gives a way to pass data from one component to other components. It is similar to function arguments. Props are passed to the component in the same way as arguments passed in a function.

Props are immutable so we cannot modify the props from inside the component. Inside the components, we can add attributes called props. These attributes are available in the component as this.props and can be used to render dynamic data in our render method.

When you need immutable data in the component, you have to add props to reactDom.render() method in the main.js file of your ReactJS project and used it inside the component in which you need.

2.3 React States

The state is an updatable structure that is used to contain data or information about the component. The state in a component can change over time. The change in state over time can happen as a response to user action or system event. A component with the state is known as stateful components. It is the heart of the react component which determines the behavior of the component and how it will render. They are also responsible for making a component dynamic and interactive.

A state must be kept as simple as possible. It can be set by using the setState() method and calling setState() method triggers UI updates. A state represents the component's local state or information. It can only be accessed or modified inside the component or by the component directly. To set an initial state before any interaction occurs, we need to use the getInitialState() method.

Chapter 3

NEED FOR OPTIMIZATION

The states and the properties (props) of the component are the two parameters that determine when the component should re-render in the application. Whenever there is a change or when a parent passes a new property to the child, the React DOM compares the new values to the previously stored values and only re-renders if there is a difference between the two states.

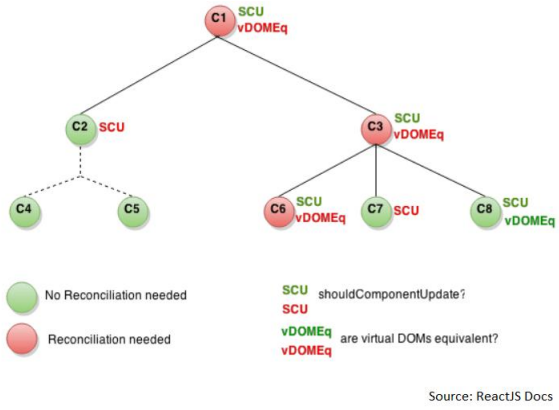


Fig. 3.1: Component Tree [1]

Consider the component hierarchy as shown in Fig 2.1. Let's say that due to some change in the component C1, the component C1 decided to re-render. Now React goes on checking the children in the subtrees of this component in a recursive manner until all the components in the subtree have been updated based on the value returned by the shouldComponentUpdate() method.

It is important to note that the components in the subtree are not forced to re-render by

the parent component and instead the re-rendering of the parent component only means that the children in that subtree will be evaluated.

From the figure, it is clear that, since SCU of C1 is true, the subtrees C2 and C3 are checked. Since SCU of C2 is false, its children are not evaluated, whereas the components in the other subtree C3 are evaluated since SCU for C3 is true and similarly the process is continued. Finally, the components C1, C3, and C6 are re-rendered.

This type of re-rendering may not cause any issues in the case of a smaller application, but in complex applications where there are a huge number of components, this re- rendering causes serious performance issues.

And relying only on the React Virtual DOM comparison may not be sufficient. Instead, some additional measures have to be taken to re-render the components only when it is required.Similar issues can only be solved by manually optimizing the react application.Hence it is necessary to optimize a react application.

Chapter 4

OPTIMIZATION TECHNIQUES

4.1 Reducing the number of State and Prop variables

This is an obvious measure to be taken. By reducing the number of State and Prop variables, the chances of the unnecessary re-renders of the component can be reduced. Also avoiding frequent and unnecessary changes to the State and the Props can be helpful. The state has to updated immediately only if has a visual impact on the user. Else the state has to be updated only at the end.

Another thing that can be done is to prevent the component from rendering is to wait until all of the essential data has been received by the component. This can be done by overriding the shouldComponentUpdate(props, state) method of the component. This method is responsible for deciding whether the component should render or not.

Consider a scenario where a component will receive the data by making a REST API call to the server and also some local props received from other components within the application. And the component is rendered when it receives the server. If the data from the server is received first and then the props from other components are received, then this would cause re-render. Instead, adding a check in the shouldComponentUpdate() to validate the data can avoid the problem.

4.2 Use React Fragments to Avoid Extra Tags

Using fragments reduces the number of extra tags that are included, only to fulfill the requirement of having a common parent in the React component.

In the case where a user is creating a new component, each component should have a single parent tag. Two tags cannot reside at the parent level, so there should be a common tag on the top.

To cater to this requirement, an extra tag is often added on the top of the component.Usually a div tag is used for this purpose.This extra div does not have any other purpose besides acting as a parent tag for the component. It is added only because the component cannot have two-parent tags.

```
import React from "react";

const App = () => {
    return (
        <div>
            <h1>Heading1</h1>
            <h2>Heading2</h2>
        </div>
    );
};
```

Fig. 4.1: Code without React Fragment [6]

To resolve the problem, the elements is enclosed inside a fragment.The fragment does not introduce any extra tag to the component but it still provides the parent to the two adjacent tags, so that the condition of having a single parent at the top level of the component is satisfied.

```
import React from "react";

const App = () => {
    return (
        <>
            <h1>Heading1</h1>
            <h2>Heading2</h2>
        </>
    );
};
```

Fig. 4.2: Code using React Fragment [6]

4.3 Do Not Use Inline Function Definition

If an inline functions is used in a component, every time the “render” function is called, a new instance of the function is created.

When React does the virtual DOM diffing, it finds a new function instance each time, so during the rendering phase, it binds the new function and leaves the old instance for garbage collection.

So, directly binding the inline function leads to extra work on the garbage collector and new function binding to the DOM. Below is an example of the inline function in the component:

```
import React from "react";

const App = () => {
    return (
        <div>
            <h1>Welcome Guest</h1>
            <input
                type="button"
                onClick={(e) => {
                    this.setState({ inputValue: e.target.value });
                }}
                value="Click For Inline Function"
            />
        </div>
    );
};
```

Fig. 4.3: Inline function code [6]

The function above creates the inline function. Every time the render function is called,

the new instance of a function is created and the render function binds the new instance of the function to the button. Also, the last function instance is available for the garbage collection, thus increasing a lot of effort for the React application. Instead of using the inline function, create a function inside the component instead and bind the event to that function itself. Doing so will not create a separate function instance every time render is called.

```
import React from "react";

const App = () => {

    setNewStateData = (event) => {
        this.setState({
            inputValue: e.target.value,
        });
    };

    return (
        <div>
            <h1>Welcome Guest</h1>
            <input
                type="button"
                onClick={setNewStateData}
                value="Click For Inline Function"
            />
        </div>
    );
};
```

Fig. 4.4: Optimizied code without inline function [6]

The above code shows the correct format for defineing a function.

4.4 Avoid Async Requests in componentWillMount()

componentWillMount will be called right before the component is rendered. This function is not used a lot. It can be used to configure initial configuration for the component but this can be done with the constructor method itself. The method cannot access the DOM elements as the component is still not mounted. Although, some developers think that this is the function where the async data API calls can be made. However, there is no benefit to this. As the API calls are asynchronous, the component does not wait for the API to return data before calling the render function. So, the component is rendered without any data in the initial rendering.

So, initially, the component is rendered with empty data and later the data is retrieved, setState is called, and the component is re-rendered. There is no big benefit to making AJAX calls in the componentWillMount phase.

4.5 Avoid Using Inline Style Attribute

With inline styles, the browser spends a lot more time scripting and rendering. A lot of time is spent on scripting because it has to map all the style rules passed to the actual CSS properties, which increases the rendering time for the component.

```
import React from "react";

const App = () => {
    return (
        <>
            <b style={{ backgroundColor: "blue" }}>Welcome to Sample Page</b>
        </>
    );
};
```

Fig. 4.5: code with inline style attribute [6]

In the component created above, there is inline styles attached to the component. The inline style added is a JavaScript object instead of a style tag.

The style backgroundColor needs to be converted into an equivalent CSS style property and then the style will be applied. Doing so involves scripting and performing JavaScript execution.

The better way is to import the CSS file into the component.

4.6 Optimize Conditional Rendering in React

Mounting and unmounting React components are costly operations. To ensure better performance of the application, the number of mounting and unmounting operations should be reduced.

There are multiple situations where, there should be a conditional rendering of the components, on the condition that may or may not render specific elements.

```
import React from "react";

import AdminHeaderComponent from "./AdminHeaderComponent";
import HeaderComponent from "./HeaderComponent";
import ContentComponent from "./ContentComponent";

export default class ConditionalRendering extends React.Component {
    constructor() {
        this.state = {
            name: "Mayank",
        };
    }

    render() {
        if (this.state.name == "Mayank") {
            return (
                <>
                    <AdminHeaderComponent></AdminHeaderComponent>
                    <HeaderComponent></HeaderComponent>
                    <ContentComponent></ContentComponent>
                </>
            );
        } else {
            return (
                <>
                    <HeaderComponent></HeaderComponent>
                    <ContentComponent></ContentComponent>
                </>
            );
        }
    }
}
```

Fig. 4.6: code with unoptimized conditional rendering [6]

In the code above, there is a conditional statement where the component is rendered according to the condition specified. If the state contains the name value Mayank, the AdminHeaderComponent is not rendered. The conditional operator and if else condition seems to be fine but the following code has a performance flaw. Let's evaluate the code above. Each time the render function is called and the value toggles between Mayank and another value, a different if else statement is executed. The diffing algorithm will run a check comparing the element type at each position. During the diffing algorithm, it seems that the AdminHeaderComponent is not available and the first component that needs to be rendered is HeaderComponent. React

will observe the positions of the elements. It seems that the components at position 1 and position 2 have changed and will unmount the components. The components HeaderComponent and ContentComponent will be unmounted and remounted on position 1 and position 2. This is ideally not required, as these components are not changing, but they are still unmounted and remounted.

This is a costly operation. The following code can be optimized like this:

```
import React from "react";

import AdminHeaderComponent from "./AdminHeaderComponent";
import HeaderComponent from "./HeaderComponent";
import ContentComponent from "./ContentComponent";

export default class ConditionalRendering extends React.Component {
    constructor() {
        this.state = {
            name: "Mayank",
        };
    }

    render() {
        return (
            <>
                {this.state.name == "Mayank" && (
                    <AdminHeaderComponent></AdminHeaderComponent>
                )}
                <HeaderComponent></HeaderComponent>
                <ContentComponent></ContentComponent>
            </>
        );
    }
}
```

Fig. 4.7: code with optimized conditional rendering [6]

In the above code, when the name is not Mayank, React puts null at position 1. When the DOM diffing occurs, the element at position 1 changes from AdminHeaderComponent to null, but the components at position 2 and position 3 remain the same. As the elements are the same, the components are not unmounted, hence reducing Unmounting and Mounting of components in the application.

4.7 Immutable Data Structures for Components

React is centered around and focused on functional programming. The state and props data in the React component should be immutable if a component must work consistently. The mutation of objects can result in inconsistent output. Let's see the code below to understand:

4.8 Using a Unique Key for Iteration

When a list of items is rendered, a key should be added for the items. Keys help in identifying the items that have been changed, added, or deleted. Keys give a stable identity to the element. A key should be kept unique for each element on the list. If the developer does not provide the key to the element, it takes an index as default key. Using index as a key will resolve the problem of maintaining the unique identity for the component, as the index will uniquely identify the component that is rendered. Index can be as key in the following scenario:

- The list items are static, items do not change with time.
- The Items have no unique IDs.
- The List is never reordered or filtered.
- Items are not added or removed from the top or the middle.

4.9 Multithreading

JavaScript is a single-threaded application to handle all the synchronous executions. While a web page gets rendered, it needs to perform multiple tasks: Handling UI interactions, handling response data, manipulating DOM elements, enabling animations, etc. All these tasks are taken care of by a single thread. Workers can be seen as an option to reduce the execution load on the main thread. Worker threads are the background threads that can enable the user to execute multiple scripts and JavaScript execution without interrupting the main thread. Whenever there are long executing tasks that require a lot of CPU utilization, those logical blocks

can be executed on the separate thread using workers. They execute in an isolated environment and can interact with the main thread using inter-process thread communication. Meanwhile, the main thread can take care of the rendering and DOM manipulation tasks.

4.10 Using Lazy Loading of Components

Bundling is the process of importing and merging multiple files into a single file so that the application dosen't have to import a lot of external files. All the main components and the external dependencies are merged into a single file and sent over the network to have the web application up and running. This saves a lot of network calls, but also leads to a problem where this single file becomes a large file and consumes lots of network bandwidth. The application keeps waiting for this large file to be loaded and executed, so delays in transmission of this file over the network lead to higher rendering time for the application. To resolve this problem,the concept of code splitting is incorporated . The concept of code splitting is supported by bundlers like webpack which can create multiple bundles for the application and which can be dynamically loaded at runtime. Loading on runtime reduces the size of the initial bundle that is created. Bundles are broken down in a way so that the components that are not loaded initially in the application can be deferred to be loaded later when they are required. This will reduce the size of the main bundle and reduce the load time of the application. Suspense and lazy can be used for this.

```
import React, { lazy, Suspense } from "react";

export default class CallingLazyComponents extends React.Component {
  render() {

    var ComponentToLazyLoad = null;

    if(this.props.name == "Mayank") {
      ComponentToLazyLoad = lazy(() => import("./mayankComponent"));
    } else if(this.props.name == "Anshul") {
      ComponentToLazyLoad = lazy(() => import("./anshulComponent"));
    }
    return (
        <div>
            <h1>This is the Base User: {this.state.name}</h1>
            <Suspense fallback={<div>Loading...</div>}>
                <ComponentToLazyLoad />
            </Suspense>
        </div>
    )
  }
}
```

Fig. 4.8: code of lazy loading [6]

In the code above, there is a conditional statement that looks for the props value and, according to the condition specified, loads either of the two components in the main component. Loading both these components initially in the main bundle will increase the overall size of the bundle. At any moment of time, one of the two components is required to be rendered. So, loading all the components that may or may not be added in the view will be a performance degrade. The components can be lazily loaded when required and these components are part of a separate chunk, loaded at runtime, which enhances the overall performance of the application. Let's understand this with the help of another example. Let's assume that there are two different components that are rendered on the basis of whether the user is logged in or not. One of the two components is rendered: WelcomeComponent or GuestComponents, depending on whether the user is logged in or not. Rather than loading both the components in the initial bundle file, let's delay the loading of the component on the basis of the condition later.

```
import React, { lazy, Suspense } from "react";

export default class UserSalutation extends React.Component {
    render() {
        if (this.props.username !== "") {
            const WelcomeComponent = lazy(() => import("./welcomeComponent"));
            return (
                <div>
                    <Suspense fallback={<div>Loading...</div>}>
                        <WelcomeComponent />
                    </Suspense>
                </div>
            );
        } else {
            const GuestComponent = lazy(() => import("./guestComponent"));
            return (
                <div>
                    <Suspense fallback={<div>Loading...</div>}>
                        <GuestComponent />
                    </Suspense>
                </div>
            );
        }
    }
}
```

Fig. 4.9: code of conditional lazy loading [6]

In the code above, both the components can be preloaded using the import keyword on the top of the component, but, instead of preloading the components WelcomeCompoment and GuestComponents, there is a conditional check. If the user name exists or not, and on the basis of the condition specified, the required component is loaded as a separate bundle. So that the initial bundle will not contain both of these components, a new bundle will be loaded on the fly according to the condition specified.

Chapter 5

CONCLUSION

React is a great framework having its own way to tackle the performance issues that the Web Applications commonly face. But let alone, the performance can still be degraded when designing complex applications where the application has to deal with a lot of data processing. This leads to the application not being responsive and application lag. This is a common problem in Enterprise Applications. This paper proposes some practical ways of tackling such problems within the web application.

The methods described aim to eliminate the redundant computations and optimize the performance of the application.

While some of the techniques described are specific to the React framework, other techniques like multithreading are related to the performance of the application itself and can be implemented in any of the application, irrespective of the framework.

REFERENCES

[1] ”Optimizing Performance ReactJS Docs” [online]. Available: *http://reactjs.org/docs* Accessed on : Nov 8 2020

[2] ”ReactJS Tutorial” [online], Available: *https://www.javatpoint.com/reactjs-tutorial* Accessed on : Nov 8 2020

[3] “AngularJS Performance Tunning for Long Lists”[online]. AvailableL: *http://tech.small-imporvements.com* Accessed on : Nov 8 2020

[4] “Web Workers API MDN Docs” [online]. Available: *developer.mozilla.com* Accessed on: Nov 8 2020

[5] “Performance Optimization Techniques for React Apps” [online]. Available: *https://www.codementor.io/blog/react-optimization-5wiwjnf9hj* Accessed on: Nov 8 2020

[6] ”React Performance Optimization Techniques” [online]. Available: *https://medium.com/technofunnel/https-medium-com-mayank-gupta-6-88-21-performance-optimizations-techniques-for-react-d15fa52c23491827* Accessed on : Nov 10 2020

[7] ”Performance Optimization Techniques In React” [online]. Available: *https://levelup.gitconnected.com/performance-optimization-techniques-in-react-31dee64c3b5* Accessed on : Nov 10 2020