IJIRCCE

e-ISSN: 2320-9801 | p-ISSN: 2320-9798

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

Volume 12, Issue 6, June 2024

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 8.379

International Journal of Innovative Research in Computer and Communication Engineering

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.379 | A Monthly Peer Reviewed & Referred Journal |

|| Volume 12, Issue 6, June 2024 ||

| DOI: 10.15680/IJIRCCE.2024.1206077 |

Optimization in React JS

Prajwal Dule, Sonali Ajankar

MCA Student, Department of MCA, VJTI Matunga, India

Assistant Professor, Department of MCA, VJTI Matunga, India

ABSTRACT: Among the challenges of starting a new React project, the overload that beginners face in the modern web development world is one of the most common problems. Frameworks, such as CRA and Vite, are supposed to allow developers to set up new projects easier than with complicated configurations, but they add more than enough useless libraries and create excessively convoluted project trees. These default settings can be confusing, as it creates clutter in the form of unopened libraries and then there are those initialized in their import history that are not necessarily relevant most of the time to new users. This document proposes a new solution: A Sample CLI tool that is tailored to provide a quality and simple boilerplate for creating an application using React. In contrast to CRA and Vite, it only has several essential dependencies that are often employed in the contemporary frontend development world: this is the main difference with Styled Components. Thus, this CLI tool, along with leaving out unnecessary packages and providing an easy to navigate structure, will make it easier for new developers to start using React for developing their applications and will make their early experience with the framework better. In this document, by detailed comparisons and problem analysis of the current popular solutions, this document shows the limitations and issues and how this custom CLI tool solves these issues and makes the process of getting started with React projects more efficient and easier.

KEYWORDS: ReactJS, Optimization, CLI, Styled-Components.

I. INTRODUCTION

As the field of web development is constantly progressing, the decision for tools and frameworks to be used can be rather influential to the overall results. It is worth mentioning that, at the present time, Create React App (CRA) and Vite have received widespread attention among developers and are the tools for beginners or experienced programmers to initiate new React projects. Although these applications facilitate the initial setup of the project, they come with a high degree of dependency on external services and a complex project architecture for the untrained eye. These issues can make studying one of the most complex tasks, thus requiring time to master and could slow a person down. There are often situations when developers face difficulties connected with the recognition and handling the so called 'crap' that is located in the root of the project, concerned fans of the liberal application of which can prevent application developers from building efficient, clear, and easily scalable JavaScript frameworks;

Being aware of this difficulty, the study concerns the creation of a new CLI tool serving to establish React applications. This tool is intended created a boilerplate with barebones and necessary dependencies such as the Styled Components needed for creating a good foundation for the application. Hence, by allowing only the basic number of pre-installed dependencies needed by most developers, this CLI tool reduces the number of unnecessary steps in set-up, confusion about how their projects ought to be configured, and enables them to work on the actual product instead of configuring it. We want to bring you a solution that is immediately recognizable as the standard while also improving the situation where you are right now, whether you are a newcomer just joining the thriving world of React and modern front-end application development.

II. METHODOLOGY

The development of our custom npm package for optimizing React applications was driven by some core objectives. Some of them include Identifying and removing some unwanted libraries and integrating important ones, making new and simple structures for easy understanding, especially for novices in the programming languages. This section explains how we systematically achieved the objectives outlined above to provide a clear understanding of the methodology employed.

We also explained how one could find those libraries the solution had generated and written unnecessary libraries that should be removed.

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.379 | A Monthly Peer Reviewed & Referred Journal |

|| Volume 12, Issue 6, June 2024 ||

| DOI: 10.15680/IJIRCCE.2024.1206077 |

a. Initial Analysis:
The first step entailed research on different React projects to examine and identify pre-existing libraries that are often incorporated. In this process, we evaluated CRA and Vite, to organize the list of libraries most often included in project solutions. It was important for us to gather some sort of initial data to build it upon in our subsequent evaluations.

After that, we looked at the usage metrics of these libraries and found the following. We analyzed their usage frequentation and the real active participation in the application's utility and efficiency. Libraries that seemed to contribute negligible additional value or where they were seldom utilized were flagged as potential candidates for elimination.

The TOP libraries in CRA and VITE were
1. React

2. Babel

3. Jest

4. Webpack

5. Typescript

6. ESlint

b. Criteria for Removal:
As for making the extractions as methodical and efficient as possible, prerequisites were set. One of them was the degree of the impact. Specific changes that created a large impact on the bundle when no noteworthy improvements resulted were perfect to be cut off.

We also aimed at getting feedback as well as comments from a beginner developer. This input enabled us to discover which libraries tended to be considered puzzling, redundant, or not important. We also added new libraries that are necessary and easy for developers.

The TOP libraries to be flagged in CRA and VITE were
1. React -necessary

2. Babel – necessary

3. Jest - unnecessary

4. Webpack – necessary but we will be using an alternative

5. Typescript – unnecessary

6. ESlint - unnecessary

c. Implementation:
Once we determined the libraries for removal, we ensured that such libraries were not included in the newly developed node package manager (NPM).

The next phase was to find and include libraries that are most important for implementing certain basics of react applications. The libraries to be used were chosen on the following criteria. We included the necessary packages for the libraries like React, React-Dom, Parcel, Styled Components for styling. These were chosen out of many due to their relevance and the fact that many people use them.
d. Integration:
We included them as pre-configured in the npm device to make it convenient to incorporate any of these essential libraries effortlessly as shown in **figure 1**. This setup comprised out-of-the-box choices that were already optimized for applying best practices, thus reducing the work that various developers have to do while getting started. The rationale

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.379 | A Monthly Peer Reviewed & Referred Journal |

|| Volume 12, Issue 6, June 2024 ||

| DOI: 10.15680/IJIRCCE.2024.1206077 |

for this was a desire to provide a turnkey solution that would facilitate the integration of WMS into other systems to improve standards compliance.

```
{
  "name": "create-boiler-plate-by-pbd",
                                                    "version": "15.0.0",
  "description": "A generator for setting up a basic React app with Parcel bundler",
  "main": "index.js",
  "bin": "bin/cli.js",
  "scripts": {
    "start": "node index.js",
    "build": "parcel build index.html",
    "install-react": "npm install react react-dom",
    "install-styled-components": "npm install styled-components"
  },
  "author": "Prajwal Dule",
  "license": "ISC",
  "devDependencies": {

    "parcel-bundler": "^1.12.5",
    "react": "^18.3.1",
    "react-dom": "^18.3.1",
    "styled-components": "^6.1.11"
  }
}
```

Figure 1: Package.json of generator file . This includes libraries that need to be included in the new CLI tool

Simplifying the Code Structure

Lastly, an attempt was made to provide a generalization of code structure to reduce obstacles for people who begin studying programming language. To define our approach to the development of the boilerplate code structure, we came up with the idea of creating a simplified and lightweight application template that contains only the necessary modules and configurations. Beyond this, We used this boilerplate as a base which developers could further work off from to kick start the application instead of going round in a new coming React application.
We also focused on the fact that it is clear where the files are saved and what folders can be found inside the Componants folder as shown in **figure 2.** In this way, organizing files logically and algorithmically helped motivate novices with the idea of the project arrangement. It was used to make the codebase more accessible and to offer direct ways of teaching.

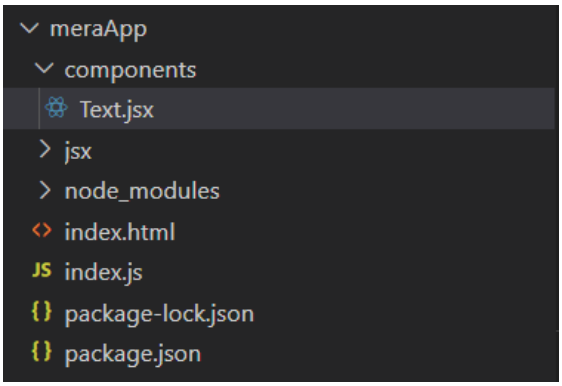


Figure 2: File Structure for newly created app by using our CLI tool

International Journal of Innovative Research in Computer and Communication Engineering

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.379 | A Monthly Peer Reviewed & Referred Journal |

|| Volume 12, Issue 6, June 2024 ||

| DOI: 10.15680/IJIRCCE.2024.1206077 |

III. IMPLEMENTATION

This section gives a glance at what we have created as a custom npm package to enhance the React applications by eliminating other unwanted variables and including the necessary ones in a simpler manner. The primary goal, therefore, is to enhance the codebase readability for entry-level programmers. The subsequent sub-processes elaborate the procedures for implementing the subsequent package, the features of this package, and the optimization strategies used.

These key steps are as follows:

1. Initializing the Project and Setting Up Dependencies

a. Creating a package. json :

- First, we ran the `**npm init**` command to kick-start the NPM package and generated our `package.json` file. to control dependencies and scripts you need to create for your application.
- Other dependencies include; React, React-Dom, Parcel and styled components these were the most important ones to include in the initial structure of the application.

b. Adding Dependencies :

- Included React styled componants and parcel so that the application would have the necessary functionalities it needs to operate on a basic level shown in **figure 3.**
- Integrated Styled Components for every component to provide a plain and efficient method of styling a component without the inclusion of several other complicities.

```
{
 "name": "create-boiler-plate-by-pbd",
 "version": "15.0.0",
 "description": "A generator for setting up a basic React app with Parcel bundler",
 "main": "index.js",
 "bin": "bin/cli.js",
 "scripts": {
  "start": "node index.js",
  "build": "parcel build index.html",
  "install-react": "npm install react react-dom",
  "install-styled-components": "npm install styled-components"
 },
 "author": "Prajwal Dule",
 "license": "ISC",
 "devDependencies": {

   "parcel-bundler": "^1.12.5",
   "react": "^18.3.1",
   "react-dom": "^18.3.1",
   "styled-components": "^6.1.11"
 }
}
```

Figure 3 : Included Libraries in package.json of Generator

2. Syntactic and Semantic Setup of the Bin Directory and Script

- Made a `bin` directory to hold the initial executing script to create the boilerplate staring with **#!/usr/bin/env node** shown in **figure 4**.
- Supplemented the discussed package with a command to the link of the executable script so that the package can be run directly from the command by the users

- Created a generator script under the `bin` directory concerning the React boilerplate.
- The script starts preparing a basic project architecture with folders for the source code folder `src`, React components folder `components`, and a styles folder for CSS or styled components shown in **figure 5.**

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.379 | A Monthly Peer Reviewed & Referred Journal |

|| Volume 12, Issue 6, June 2024 ||

| DOI: 10.15680/IJIRCCE.2024.1206077 |

```
#!/usr/bin/env node
```

Figure 4

```
const boilerplateFiles = [
   // Add Parcel configuration files
   { filename: 'index.html', content: `<!DOCTYPE html>
<html>
<head>
   <title>My App</title>
</head>
<body>
   <div id="root"></div>
   <script src="index.js"></script>
</body>
</html>` },

   { filename: 'index.js', content: `import React from 'react';
import ReactDOM from 'react-dom';
import App from './jsx/App.jsx';
ReactDOM.render(<App/>, document.getElementById('root'));` },

   { folder: 'jsx', filename: 'App.jsx', content: `import React from 'react';
import styled from 'styled-components';
import Text from '../components/Text';

const Container = styled.div\`
  font-size: 2em;
  text-align: center;
  display:flex;
  align-items: center;
  color: palevioletred;
  background-color:black;
\`;

function App() {
  return (
    <Container>
      <Text />
    </Container>
  );
}

export default App;` },

   { folder: 'components', filename: 'Text.jsx', content: `import React from 'react';
import styled from 'styled-components';

const StyledText = styled.p\`
  font-size: 1.5em;
  color: blue;
\`;

function Text() {
  return (
```

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.379 | A Monthly Peer Reviewed & Referred Journal |

|| Volume 12, Issue 6, June 2024 ||

| DOI: 10.15680/IJIRCCE.2024.1206077 |

```
    <StyledText>Hello World </StyledText>
  );
}

export default Text;` },

    { filename: 'package.json', content: `{
  "name": "",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "start": "parcel index.html",
    "build": "parcel build index.html"
  },
  "dependencies": {
    "react": "^17.0.2",
    "react-dom": "^17.0.2",
    "styled-components": "^5.3.3"
  },
  "devDependencies": {
    "parcel-bundler": "^2.0.0"
  }
}` }
];
```

Figure 5

3. Simplifying the Boilerplate

- Merged features that can be seen in other templates and are not so relevant and useful, such as elaborate welcome pages or counter examples.
- Offered an uncomplicated and clear “Hello, World!” message on the main page as the reference point for novices.
- Made certain that the project would have readable folder and file names and organization that would follow industry standards.
- I also provided sample files for Styled Components so that the style of the React components can be developed without burdening new developers shown in **figure 5**.

4. Publishing the Package

- Had to sign up for an NPM account in order to publish the package.
- Set the package to be tied to the **npm login** so that the other developers could install it.
- Employed the NPM command: **npm publish** to publish the package to the npm registry.
- Ensured that clear instructions on how to install the package were well announced on the npm page and subsequent usage examples.

5. The package can be used to develop a boilerplate.

- It is always possible to install the package globally by using the following command in the terminal: **`npx create-boiler-plate-by-pbd myapp`**.
- This package can be run with just as a command (for example, ` create-boiler-plate-by-pbd `) that calls the generator script and creates the basic project structure.
- The resulting boilerplate intentionally is simple and minimalistic, which makes it easy to add extra features and libraries for developers.
- Offered advice on how to increase the size of the project and ways to write clean code that could be considered as best practice.

To sum up, our custom npm package creates React applications with less boilerplate code, while remaining minimal and well-optimized.

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.379 | A Monthly Peer Reviewed & Referred Journal |

|| Volume 12, Issue 6, June 2024 ||

| DOI: 10.15680/IJIRCCE.2024.1206077 |

IV. RELATED WORK

In trying to improve the efficiency of React application performance the 'Don't Render When You Don't Need To' is a great rule of thumb. This can be done using some strategies like memoization and lifecycle management of components. One of the efficient solutions is the method using React's useCallback hook, described by Zhang Leng in the thesis [2]. Leng shows an example to explain how useCallback assists in the process of having fewer function references when required by preventing the function from getting new references on every re-rendering unless the dependency list has a change. Since it prevents restructuring of references to functions, useCallback minimizes unnecessary renders that pose a serious threat to the efficiency of the React applications. It is most helpful if the components contain code that performs computationally expensive calculations or complex rendering that should not be duplicated for each instance.

The second useful hook aimed at component optimization is called useMemo. According to Roy Anal [1], you could utilize the useMemo utility in editing and memoizing the results of parameters with giant calculations or objects to guarantee that it will be updated in line with the change of its parameters. This hook is especially useful since it helps to optimize functional components where it is possible that the calculation of some value is rather power-consuming and needs to be performed only if such a need arises. For instance, if a component computes a value/derived data, useMemo will be useful to cache the value and return the cached value as the subsequent renders occur so long as the dependency array does not change. This makes it easier in curtailing down the number of computations that are done, and consequently enhances the efficiency of the component.

Unlike React. memo, which to avoid the unnecessary re-rendering of a component based on its props, the Same for useMemo which is for avoiding the re-computation of determined values or expressions within a component. Because this approach is specific to the component, developers can also set very specific performance optimizations and minimize the number of renders happening in a tree that contains components.

The first and foremost step towards successfully optimizing React applications is the knowledge of profiled performance bottlenecks. The gained knowledge should be applied to avail this or that capability, which is explained by the author of the work "Optimization in React.js" Iida Kainu [3]. The React Profiler is also beneficial to developers as it provides rendering time breakdowns of components helps developers to identify which component re-renders too often and which one requires optimization. Kainu also stresses the need to draw some conclusions about the application of performance improvements based on profile data. For example, it is possible to analyze the real cases similar to those described by Zhang when MemoedIncrement and MemoedMultiply components react to parent updates In this case, the React Profiler can clearly see that when possible, stable function references, protected by useCallback, enhance their re-renders. These visualizations make things more real for developers by showing them the real-world effects of their optimization and guiding their subsequent work on optimization.

When speaking of valuable addition in terms of creating the web-app boilerplates, we must mention the approach based on Leopold's work in [4] on the custom React app generator. Leopold's tool will again be similar to CRA tool with most of the work being handled during the set up for the website developers but will be more flexible and also more modern than CRA in as much as it will fulfill the individual's needs. This generator lets the developers avoid any dull setup typical to new projects, and it provides patterns along with vital configurations and dependencies that are a must-have for a React project.

V. RESULTS

This section demonstrates and discusses how our custom NPM package influences the specified React development process for the beginners. Although the scope of the changes wasn't on actual performance gains, the package greatly improves the usability, setup time and overall developer experience. However, the first time it takes a relatively longer time to load down since there is a initial setup and this might take a relatively longer time; but in the long run this system is very simple and the load is relatively very small.

1. Developer Productivity

• **Before:** Newcomers to development tend to struggle with several challenges because they need to learn different tools and libraries that they have to use to compile working programs.

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.379 | A Monthly Peer Reviewed & Referred Journal |

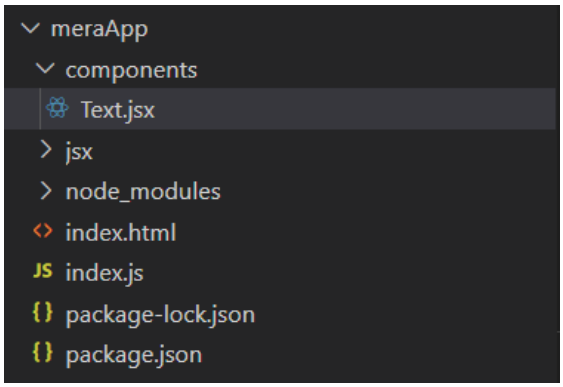
|| Volume 12, Issue 6, June 2024 ||

| DOI: 10.15680/IJIRCCE.2024.1206077 |

• **After:** The package is well structured and comes with neatly designed square brackets which enable a developer to start coding. Some of the responses we received from the users of this software and tool were about the time taken to construct the first component even though they reported longer time in loading the application to start with it.
•**Before:** People writing unnecessary boilerplate code may overwhelm beginners with excessive information not related to core React concepts.
• **After:** Although the package is very simple, it allows developers to start from the basics and learn how React works as a basic concept. This facilitates in reinforcing the basic fundamental knowledge in a more effective manner through this targeted approach.

2. Code and Projects are grouped to enhance understanding and to reduce complexity.
a. Intuitive Project Structure:
• **Before:** The first issue that a beginner may face is the organization of the projects and sub-projects.
• **After:** The package creates a logical and easy-to-follow structure for the project based on importing required packages. .
b. Streamlined Dependencies:
•**Before:** Multiple dependencies are challenging when implemented by fresh developers and are potentially special nuisances.
•**After:** Some of the modules included are; React and Styled Components only which enhances the only dependency management and reduces chances of errors.

3. User Experience Improvements
a. Simplified Setup Process:
•**Before:** New developers in the React environment have problems with the initial setup of a React app, the presence of numerous complicated configurations and numerous choices of libraries.
• **After:** The custom NPM package eliminates most of the setup process, leaving only one command to execute and setting up a bare-bone structure that is barely necessary.
b. Reduced Cognitive Load:
• **Before:** Some of the React boilerplates contain a large amount of example code and multiple dependencies, which may be easily disorienting to new developers.
• **After:** Since the package only outputs a “Hello World!” message and imports solely the necessary libraries, the user is not overwhelmed when beginning with the program. The new writers mentioned that they experienced a lower level of confusion when introduced to code, signifying ease for novices.



Text.jsx

```
import React from 'react';
import styled from 'styled-components';

const StyledText = styled.p`
  font-size: 1.5em;
  color: white;
`;
```

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.379 | A Monthly Peer Reviewed & Referred Journal |

|| Volume 12, Issue 6, June 2024 ||

| DOI: 10.15680/IJIRCCE.2024.1206077 |

```
function Text() {
  return (
    <StyledText>This is a text component.</StyledText>
  );
}

export default Text;
```

Output:

VI. CONCLUSION

In conclusion, with the goals of this research, this paper has sought to improve the experience of the new developers in React. From the analysis of libraries that are frequently used in the process of constructing applications, together with supplementary and detailed research focused on leaving out any libraries that are not considered crucial for the process, this research has proved to be very useful in enhancing the process. The recent launch of a new CLI tool of working on the project means it uses only required packages, which makes the learning process easier and contribute to the use of effective React development practices.

The future work should go on focusing the reduction of time required for the installation of the CLI tool. Though in the current tool, the library selection and usability are taken into account, the initial tool-setting time is still a big challenge. Increasing installation efficiency would not only make installations look better to the end user but also attract more developers to use this tool or build projects in react, ensuring a diverse group of people dedicated to the efficient and intelligent development of projects using React.

Furthering research on tools such as this CLI can unveil better options for developers to augment their embrace of React at all years of efficiency, improving its readied resurrection to modern web development.

ACKNOWLEDGMENT

It is my great pleasure to acknowledge the following authors for their contribution in the subject of React optimization and performance evaluation – Anal Roy, Zhang Leng, Iida Kainu. Their profound research and papers have been beneficial for creating massive influence to this study.

REFERENCES

1. Roy Anal , " React JS for Web Developers ", CENTRIA UNIVERSITY OF APPLIED SCIENCES, 2021.
2. Leng Zhang, " REACT APPLICATION OPTIMIZATION ", California State Polytechnic University, Pomona, 2021.
3. Iida Kainu, “ Optimization in React.js: Methods, Tools, and Techniques to Improve Performance of Modern Web Applications,” Tampere University,2022.
4. Leopold, Dev Community “Generate your web-app boilerplate like create-react-app does ”, 2021

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi crossref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

9940 572 462 6381 907 438 ijircce@gmail.com

www.ijircce.com

Scan to save the contact details