National Conference on Information Technology & Digital Application (NCITSA-2025)
International Journal of Advanced Engineering Research and Science (IJAERS) Special Issue
https://dx.doi.org/10.22161/ijaers.ncitsa.2025.5 ISSN: 2349-6495(P) | 2456-1908(O)

Dynamic Rendering Using ReactJS

Ms. Deepika Bansal[1] , Daksh Jain[2] , Avani Jain[3],

[1]Assistant Professor
Department of Information Technology, Jaipur Engineering College & Research Centre, Jaipur
[2]Student
Department of Information Technology, Jaipur Engineering College and Research Centre, Jaipur
Email: dakshkain.it25@jecrc.ac.in
[3]Student
Department of Information Technology, Jaipur Engineering College and Research Centre, India
Email: avanijain.it25@jecrc.ac.in

Abstract*— Dynamic rendering is a critical technique in modern web development that enhances the performance and SEO of JavaScript-heavy websites. ReactJS, as a popular front-end JavaScript library, provides an efficient way to implement dynamic rendering by allowing developers to create reusable UI components and manage dynamic data changes seamlessly. This paper explores the integration of dynamic rendering in ReactJS applications to improve performance, ensure efficient content delivery, and enhance the user experience. The focus is on addressing the limitations of traditional client-side rendering for SEO and performance bottlenecks, especially on low-end devices or slow networks. The methodology involves building a sample application with dynamic data and analyzing performance improvements using server-side rendering (SSR) and Reacts hydration process. Key findings show that dynamic rendering offers significant improvements in load times, crawlability, and overall user satisfaction.

Keywords* — Dynamic Rendering, ReactJS, Server-Side Rendering (SSR), Hydration, SEO Optimization

I. INTRODUCTION

Dynamic rendering has become increasingly vital in the development of high-performance web applications. With the rise of single-page applications (SPAs), traditional client-side rendering often poses challenges for search engine bots and affects the loading speed of applications. This has led to a growing interest in dynamic rendering solutions that combine the strengths of client-side and server-side rendering. ReactJS has emerged as a powerful library for building user interfaces, offering the flexibility to manage both dynamic content and server-side rendering through frameworks like Next.js. The primary objective of this paper is to explore how dynamic rendering, when implemented using ReactJS, can improve application performance, SEO, and the overall user experience. The discussion includes a comparative study of rendering techniques, highlighting the advantages of dynamic rendering and its applicability in modern web development scenarios.

II. METHODOLOGY

The research methodology follows a structured approach involving application development, server-side integration, and comprehensive evaluation and testing. This multi-layered process was designed to assess the technical and experiential benefits of dynamic rendering using ReactJS.

A. Application Development

The initial step involved developing a prototype ReactJS web application that mimics a real-world dashboard. This application featured dynamically generated content including live user data fetched from an API, interactive data visualizations using chart libraries like Chart.js, and seamless navigation through React Router. The architecture was component-based, emphasizing modularity and reusability. The application was initially

https://dx.doi.org/10.22161/ijaers.ncitsa.2025.5 ISSN: 2349-6495(P) | 2456-1908(O)

rendered entirely on the client side. This meant that the HTML was mostly blank on page load, and all content was injected dynamically via JavaScript after the browser downloaded the JavaScript bundle and executed it. While this approach offers a snappy user experience once loaded, it suffers from slow first paint and SEO limitations, particularly because search engine bots may not fully execute JavaScript to render the content.

B. Server-Side Integration

To overcome the limitations of client-side rendering, the application was restructured using Next.js, a popular React framework that supports Server-Side Rendering (SSR) out of the box. SSR enabled the application to send fully rendered HTML from the server to the client, making the initial content visible faster and crawlable by search engine bots. Additionally, React's hydration process was employed, which involves attaching event listeners to the server-rendered HTML elements once the client-side JavaScript loads, making the page fully interactive without re-rendering from scratch. This transition required refactoring components to be compatible with SSR and managing data fetching both on the server and client. Key challenges such as managing asynchronous data, preserving global states, and ensuring smooth transitions between static and dynamic content were addressed during this phase.

C. Evaluation and Testing

Post-implementation, the application's performance and usability were rigorously evaluated. Quantitative performance metrics were collected using tools such as Google Lighthouse, WebPageTest, and PageSpeed Insights. Key metrics analysed included:

Time to First Byte (TTFB): How quickly the server responds.

First Contentful Paint (FCP): When the first bit of content appears.

Largest Contentful Paint (LCP): When the main content is fully loaded.

These metrics were compared between the client-rendered and server-rendered versions to highlight improvements. Additionally, the application was tested for SEO efficiency using Google Search Console to ensure proper indexing and crawlability of dynamic content.

On the qualitative side, user feedback was collected via usability tests and questionnaires. Testers reported on perceived load times, responsiveness, and content visibility. These insights provided valuable information about real-world usability improvements introduced by dynamic rendering techniques.

III. IMPLEMENTAION & ANALYSIS

To practically implement and assess the benefits of dynamic rendering, a full-fledged ReactJS application was developed and later optimized using Next.js. The application was designed as a feature-rich dashboard with real-time data visualizations, user authentication, dynamic routing, and API-based data updates. This setup closely resembles modern production-level applications, providing a suitable environment for evaluating rendering techniques.

The implementation process was divided into three key phases:

A. Baseline Testing

Initially, the application was deployed using traditional client-side rendering (CSR), where the browser receives a minimal HTML file and fetches all necessary content through JavaScript after the page load. This model, although popular for SPAs, revealed several drawbacks. One major issue was delayed content visibility, especially on slower connections or low-powered devices. Since the actual content only loads after the JavaScript is fully executed, users experienced blank screens for longer durations, leading to a poor first impression. Furthermore, search engines struggled to index the dynamic content because bots often cannot execute JavaScript effectively.

https://dx.doi.org/10.22161/ijaers.ncitsa.2025.5 ISSN: 2349-6495(P) | 2456-1908(O)

This affected the Search Engine Optimization (SEO) of the application, resulting in low visibility in search engine results. This phase served as the benchmark to compare improvements made through dynamic rendering.

B. SSR and Hydration

The next phase involved transitioning the application to use Server-Side Rendering (SSR) via Next.js. SSR allowed the server to send a fully-rendered HTML page for each request, which was immediately viewable by the client and easily crawlable by search engines. The application maintained full interactivity through hydration, where React's JavaScript bundle attaches event listeners and reactivates the pre-rendered HTML elements. This method preserved the dynamic behavior of the application while significantly improving initial load performance. Users noticed content appearing almost instantly, and bots could parse page content more effectively. Implementing SSR required some architectural changes, such as restructuring data-fetching logic using getServerSideProps() and getStaticProps() to preload data before rendering. This step validated the technical viability of dynamic rendering in a React environment.

C. Performance Analysis

The final step involved in-depth performance and user experience analysis. Using tools like Google Lighthouse, WebPageTest, and PageSpeed Insights, key metrics were recorded and compared between CSR and SSR implementations:

First Contentful Paint (FCP): Improved by approximately 40%, meaning content became visible faster.

Largest Contentful Paint (LCP): Reduced by about 35%, indicating faster rendering of the main content.\n- Time to Interactive (TTI): Became more consistent across devices.

SEO performance also improved notably. Google Search Console revealed faster crawl rates and successful indexing of previously unreachable content. This ensured higher visibility on search engines. From a usability standpoint, user testing sessions demonstrated increased satisfaction. Users reported a smoother experience with less waiting time and faster access to interactive components such as navigation menus and charts.

These analyses confirmed that dynamic rendering using ReactJS and Next.js not only improves raw performance metrics but also enhances the user experience, accessibility, and search engine friendliness of web applications.

VI. RESULT

The research demonstrated that dynamic rendering using ReactJS notably improves web application performance, SEO, and user engagement. By pre-rendering content on the server and hydrating it on the client, applications achieve faster load times and become more accessible to search engine crawlers. Quantitative analysis showed marked reductions in key performance metrics, and qualitative feedback confirmed better user satisfaction. Thus, incorporating dynamic rendering strategies into ReactJS applications is highly beneficial for developers aiming to optimize modern web experiences.

V CONCLUSION

This research explored the practical and performance benefits of implementing dynamic rendering in ReactJS applications, specifically through the integration of server-side rendering and hydration using Next.js. The findings clearly demonstrate that traditional client-side rendering, while flexible and widely adopted, presents limitations in performance and search engine visibility. Through structured methodology and in-depth implementation, it was observed that dynamic rendering significantly improves key performance indicators such as First Contentful Paint (FCP), Largest Contentful Paint (LCP), and Time to Interactive (TTI).

International Journal of Advanced Engineering Research and Science (IJAERS) Special Issue
https://dx.doi.org/10.22161/ijaers.ncitsa.2025.5 ISSN: 2349-6495(P) | 2456-1908(O)

Moreover, dynamic rendering ensures better SEO compatibility by enabling search engines to crawl and index dynamic content more effectively. From a user experience perspective, the shift to server-side rendered pages provided faster load times and more seamless interactions, directly enhancing overall user satisfaction.

In conclusion, dynamic rendering with ReactJS, powered by frameworks like Next.js, emerges as a robust and efficient approach to building high-performance web applications that meet both user expectations and technical SEO standards.

IV. ACKNOWLEDGMENTS

We would like to express our sincere gratitude to the faculty and mentor for their continuous support and guidance throughout this research. Their insights into ReactJS and web performance optimization were invaluable. We also extend our thanks to the users and testers who participated in our evaluations, providing essential feedback that helped shape our findings. References

REFERENCES

1. Google Developers. "Dynamic Rendering." https://developers.google.com/search/docs/crawling-indexing/dynamic-rendering
2. ReactJS Documentation. https://reactjs.org/docs/getting-started.html
3. Next.js Documentation. https://nextjs.org/docs
4. Mozilla Developer Network (MDN). "Server-Side Rendering." https://developer.mozilla.org/en-US/docs/Glossary/SSR
5. Lighthouse. https://developers.google.com/web/tools/lighthouse
6. WebPageTest. https://www.webpagetest.org/
7. Search Engine Journal. "How JavaScript Impacts SEO." https://www.searchenginejournal.com/javascript-seo-guide/
8. Smashing Magazine. "React Hydration: How It Works." https://www.smashingmagazine.com/2021/04/react-hydration/
9. Medium. "Understanding Dynamic Rendering with React and Next.js." https://medium.com/@someauthor/dynamic-rendering-react-nextjs
10. Google Search Central Blog. "Rendering on the Web." https://developers.google.com/search/blog/2019/11/rendering-on-web