Why Next.js is very slow on dev mode and How to Improve its Performance.

Dr. S. Sasikumar, B. Chandra Mohan
Hindustan Institute of Technology and Science, ECE
ssasik@hindustanuniv.ac.in, chandraroy4214@gmail.com

ABSTRACT – We know that building a complete web application involves a lot of technologies and combining all of them in one place takes a lot of time and effort of developers. That's where frameworks like Next.js come into play. Next.js bundles all of the packages and config files together in a neat and organized way. It is unique because it's full-stack web application framework, where we can write all the frontend and backend code of an application in one place. So, it makes the developer life a bit easy and helps in shipping the product as fast as possible. But the only problem with full-stack frameworks like next.js is since we have to write the whole code in one place, every time we do a production build, the compilers of next.js has to compile the whole code base. That's where I found some space for Improvement. In this article I will try to explain how can we Improve the performance of production build next.js app by using some of the techniques and coding patterns which I learnt while building web apps with next.js.

Keywords: next.js, performance, client-side rendering, server-side rendering, routing, dynamic routing, Image Optimization, page pre-rendering, lazy loading.

I. INTRODUCTION

NextJS is a popular React framework for building server-side rendering web applications. There are three key benefits to using NextJS: A more pleasant user experience. Performance that is above ordinary. The purpose of NextJS is to make your life as a React developer easier. It also enriches React by offering many key functionalities that you would otherwise have to add to React apps on your own. Routing is one example, but there are many others. NextJS extends the functionality of your React app.

React.js is a responsive system that "changes the principles of the game" with regards to web advancement; being exceptionally componentised, you split your application into little squares and those squares can contain the rationale along with the construction and even styles, bound to the part and not the entire application. This was a weighty element permitting bigger application bases to be made substantially more solid and to have less "complex components" that break.

Lately, web application advancement has gone through an extreme change because of the ascent of Next.js. This system permits engineers to fabricate strong web applications with JavaScript without stressing over building the back-end foundation. It works on the most common way of making crossover applications with client-side as well as server-side delivered pages. While the system is straightforward, engineers actually battle to speed up their applications.

An application's speed is directly connected with how much time it takes to serve the application code, styles, and information to the client in the first full circle. Whenever the server needs to send extra resources (for instance pictures) during the underlying full circle, the application execution corrupts. Luckily, designers can follow various prescribed procedures to work on the speed of their Next.js applications.

II. LITERATURE SURVEY

Next.js 9 presented getStaticProps, which permits engineers to bring information at construct time. Dissimilar to getInitialProps, this capacity generally runs server-side! The code composed inside this capacity will not be remembered for the JavaScript pack shipped off the client, so it's protected to compose server-side code straightforwardly in getStaticProps. Rather than depending on outer API endpoints to bring the information expected to deliver the models on the/store page, this information is accessible to us quickly while exploring to the page.

Image format: The <Image/> component supports the webp image format, which is the next-generation picture format. JPEG files with the same quality index are 25-35 percent smaller than images in this format.

When comparing the sizes of the fetched photos, we can clearly see the difference: whereas the red vehicle model's image used to be 1.3MB in the old method, we were able to reduce the size by -98.75 percent to only 16.6kB by using the component. The term "image optimization" refers to the process of lowering the size of a picture file. Because photos are one of the most significant components slowing down your app's performance, shrinking image files can help. It's a two-step procedure: 1) Reduce the size of the photograph and 2) save it in the proper format (jpeg is better for photos; png is better for graphics).

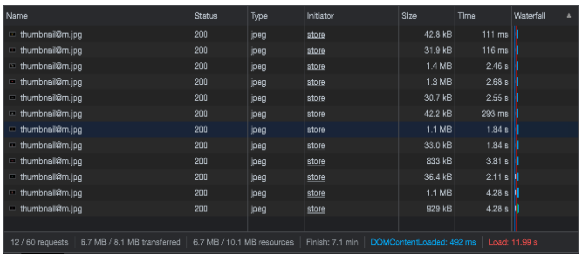


Fig-1: Statistics of the Image downloaded before Optimization

Lazy Loading: The prior implementation made retrieve requests to get all of the photos for all of the models. Only when the viewport intersects the bounding box of the picture does the Image /> component get the image. Despite the fact that no changes to the photos themselves were required, we were able to reduce the image loading time from an average of 3000ms to 270ms.

Dynamic Routes: While perusing the store, clients can tap on everything to all the more likely view the model. The old execution utilized a mix of question boundaries and getInitialProps to deliver the page and bring the required information to deliver each model. Like what we saw on the/store page, the client can see the model once the API demand started inside the getInitialProps work has settled.

Next.js 9 presented document framework based unique courses. In mix with the new getStaticPaths work utilized along with getStaticProps, this element makes it conceivable to powerfully pre-render the model pages in light of their id. Rather than having one model page and utilizing getInitialProps and inquiry boundaries to figure out which information to get and what model to deliver, we can straightforwardly utilize a way boundary to statically create pages for each model. By enveloping each model card by the network in a <Link/> part, we can prefetch each/model/[id] page once the card shows up in the viewport, permitting moment route when a client taps on the card.

File based Routing: Routing in React, in my opinion, is a terrible need. It contains a lot of moving elements and necessitates a lot of time, code, and effort to do it correctly. Next.js has a built-in mechanism for file-based routing, which means that your app's routes are decided by its file structure. The mandated file-structure organises the programme and allows developers to observe the app's navigation structure visually. This code-free way to routing is actually very intuitive, and I think it's a great idea. Each Next.js project includes a page's folder, which contains all page-level graphic components used throughout the programme.

Developer Experience: Unlike previous versions of Next.js, which required a lot of specific setting, Next.js 9 comes with TypeScript support out of the box! Instead of dealing with our own configuration, we can start utilizing TypeScript by adding a tsconfig.json file to the root of existing projects or by calling 'npx create-next-app —ts' on newly generated projects.

Client-side rendering: Client-side delivering (CSR) implies delivering pages straightforwardly in the program utilizing JavaScript. All rationale, information bringing, templating and directing are dealt with on the client instead of the server. Client-side delivering can be hard to get and save quick for portable. It can move toward the presentation of unadulterated server-delivering if accomplishing insignificant work, keeping a tight JavaScript spending plan and conveying esteem in as not many RTTs as could really be expected. Basic contents and information can be conveyed sooner utilizing HTTP/2 Server which gets the parser working for you sooner. Designs like PRPL merit assessing to guarantee introductory and resulting routes feel moment.

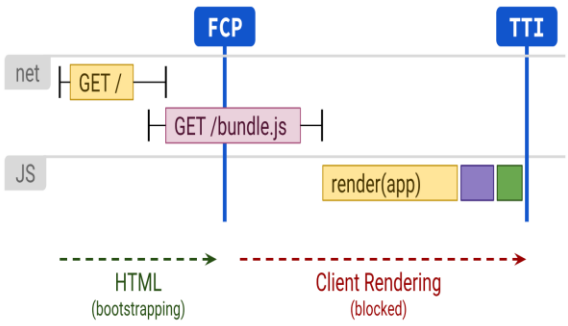


Fig-2: Serving the scripting bundles from Server

The essential drawback to Client-Side Rendering is that how much JavaScript required will in general develop as an application develops. This turns out to be particularly troublesome with the expansion of new JavaScript libraries, polyfills and outsider code, which vie for handling power and must frequently be handled before a page's substance can be delivered. Encounters worked with CSR that depend on enormous JavaScript packs ought to consider forceful code-parting, and make certain to apathetic burden JavaScript - "serve just what you want, when you really want it". For encounters with almost no intelligence, server delivering can address a more adaptable answer for these issues. Next.js 12 has a new Rust-based compiler that takes use of native compilation and is built on SWC. By simply upgrading the Next.js version, we were able to reduce our build time from 90s to 30s.

React Fast Refresh: Quick Refresh is a Next.js highlight empowered in all Next.js applications on variant 9.4 or more current. It gives immediate criticism on alters made to your React parts in somewhere around a second without losing part state. The presentation of SWC in Next.js 12 worked on the revive rate fundamentally, coming about in 3x quicker invigorates contrasted with earlier forms.

Vercel suggest that you eliminate conditions individually and restart your application after every expulsion to guarantee that the reliance was genuinely not required and that you didn't break your application.

III. PROPOSED METHODOLOGY

Image Optimization: One of the most resource-consuming parts of a web application is the handling of images. Images often present a challenge to me as a developer because they have proven to be large, bulky, and slow to process. Next.js comes packaged with a really phenomenal image component that makes the optimization for images extremely straightforward. It completely removes the need to have copies of the same images in different dimensions, qualities, and formats by handling all of these pain points behind the scenes which allows for good quality images to be loaded into the application quickly.

The size of the downloaded image is drastically decreased from 6.7MB to 134kb, this is the biggest performance booster. You can see the loading time is drastically reduced from 11.99s to 1.44s. Optimization techniques like these can really enhances the experience of the user.

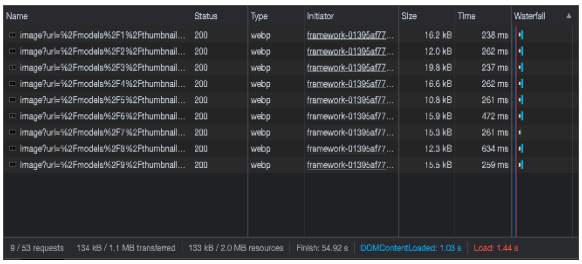


Fig-3: Statistics of the Image downloaded after the Optimization

Styling: styling patterns of the components makes or breaks the performance of the app or website. The right styling patterns like separation of concerns, using same styling classes as much as possible for unique items. Importing styling before the scripting files adds an advantage to the total performance of website or app.

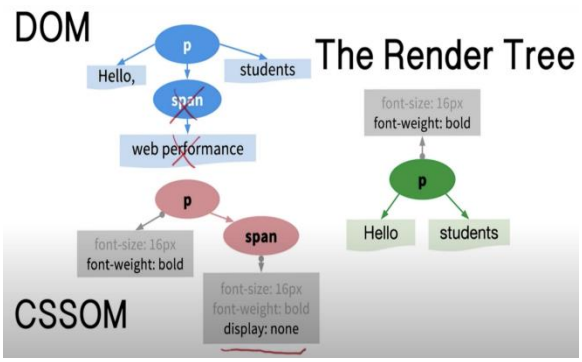


Fig-4: Tokens of Tree Structure generated from Abstract Syntax Tree

Remove Unused Dependencies: Numerous applications rely upon outsider bundles. While conditions are certainly really great for your application, they increment its size and stacking time. Assuming you are utilizing npm bundles in your Next.js application, you ought to look for unused conditions. They occupy room in your last pack and could cause unforeseen ways of behaving in your application. Assuming you have a little undertaking, you can undoubtedly track down the unused conditions and eliminate them from the package.json record of your Next.js application. Yet, assuming that you have a huge undertaking with heaps of various conditions, observing the unused dependencies might be troublesome. For this situation, utilize the depcheck bundle to track down unused conditions in your undertaking (this bundle is incorporated with npm).

Server-side rendering: When it comes to scripting files the server returns a ready-to-render HTML page as well as the JS scripts needed to make the page interactive when using JavaScript libraries like React. All static components in the HTML are rendered

immediately. The browser downloads and executes the JS code in the meanwhile, making the page interactive. The client-side interactions on this page are now handled by the browser. The browser sends an API call to the server for any new content or data, and just the newly required information is fetched. The most importantly necessity for server-side delivering is to have a runtime climate that can execute a web server and handle occasions.

Node.js is quite possibly the most well-known system to set up SSR for a React application and Express is an extraordinary choice for making the HTTP server. Then, we really want a JavaScript compiler like Babel and a JS module bundler like Webpack, Rollup or a comparative device.

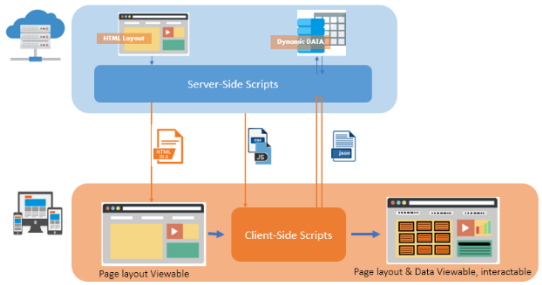


Fig-5: Render tree generated from Server-Side Rendering

Page pre-Rendering: Another component that Next.js gives out of the case is Page Pre-delivering. Page pre-delivering fills an exceptionally enormous hole found in many applications that are created utilizing the React library. Single page applications are a bad dream for site motor enhancement because of the way that they just have a solitary page for the web indexes to creep and accumulate information from. The single page is typically an extremely fundamental standard that has a solitary hub which gets filled progressively after page loads. As well as tackling this issue, pre-delivering likewise gives quicker load times and the very smooth feel that solitary page applications are known for.

There are two kinds of pre-delivering that Next.js gives: static age and server-side delivering. Static age is an age of html pages at fabricate time for the web server to serve when the given page is mentioned. Server-side delivering, then again, is the age of html documents on the server upon each solicitation that is gotten by the server. Server-side delivering is worked with by an alternate strategy, getServerSideProps. How about we investigate the two techniques that work with static age, getStaticProps and getStaticPaths.

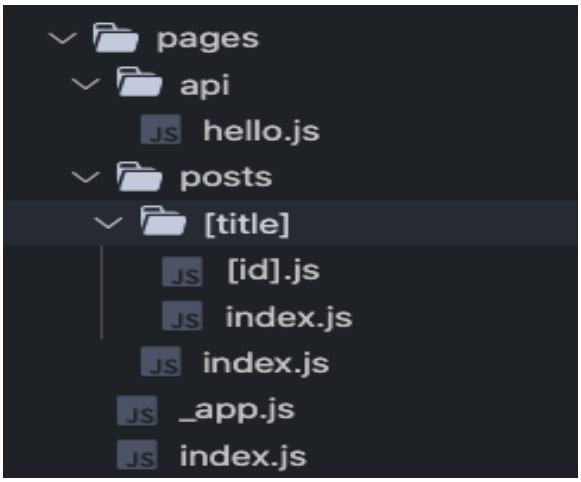
File system-based routing: Next.js has a document framework put together switch worked with respect to the idea of pages. Each document in the pages catalog is related with a course founded on its record name. For instance, if you somehow managed to make a document named about.js, then, at that point, it would be related with the '/about' course. On the off chance that you make a record named contact.js, it would be related with the '/contact' course. The index.js record is naturally the foundation of the catalog, and that implies is related with the '/' course.

There is another document named _app.js. This App part is unique since it is industrious all through all pages in the application. It tends to be utilized to share worldwide styles, format, or information through the application.

You can make settled courses utilizing a settled organizer structure. For instance, you can make a posts organizer and include different documents inside the envelope. The first-post.js document will be related with the '/posts/first-post' course. The second-post.js document will be related with the '/posts/second-post' course. Like in our root registry, the index.js document in the posts envelope will be related with the '/posts' course.

The last organizer which we have not discussed at this point is the programming interface envelope. Next.js permits you to construct your own API. Any records within this organizer will be treated as an API endpoint rather than a page. We won't dive deep into this subject in this article, yet realize that this is conceivable with Next.js.

Dynamic Routing: To deal with dynamic courses in Next.js, you can add sections to a page. Recall our posts organizer with different records? We can transform that into one powerful document utilizing square sections, [id].js.



Now, if we go to the `/posts/first-post` or `/posts/second-post` route, it will be matched by the [id].js file. We can still access the `/posts` route using the index.js file in the posts directory. You can also include nested dynamic routes by adding square brackets to a folder. In the example below, you can have a route such as `/posts/first-post/main`. You can settle however many organizers as you like, yet regularly you won't have to go further than 4 or 5 levels. The [title] envelope and [id].js document can be named anything, the length of they have the square sections. Anything we name these will be utilized as the inquiry key, which we will see an illustration of later on when we utilize the useRouter.

V CONCLUSION

A framework like next.js comes with lot of useful features and optimizations built-in, as many of the developers say Vercel team built Next.js to make the experience of building web applications a bit easier and to help shipping of the product a bit faster. But, in this paper I tried explaining my experience with next.js while building web apps and my pieces of learnings from that journey. I hope you learnt something from this journal paper and if it can make a little difference in your life postivley, it's my pleasure. I hope, in the coming updates some of these optimizations will come built-in to provide the advantage to every developer, who is trying to make this world a better place.

VI REFERENCES

[1] Yu-Chuan Huang, I-No Liao, Ching-Hsuan Chen, Tsì-Uí İk, Wen-Chih Peng " TrackNet: A Deep Learning Network for Tracking High speed and Tiny Objects in Sports Applications "

[2] Nien-En Sun Yu-Ching Lin Shao-Ping Chuang Tzu-Han Hsu Dung-Ru Yu Ho-Yi Chung "Efficient Shuttlecock Tracking Network "

[3] Shah Rutvik Vrajesh , Amudhan A N, Lijiya A , Sudheer A P " Shuttlecock Detection and Fall Point Prediction using Neural Networks "

[4] M. Leo, G. Medioni, M. Trivedi, T. Kanade, and G. M. Farinella, "Computer vision for assistive technologies," Com puter Vision and Image Understanding, vol. 154, pp. 1–15, 2017.

[5] A. Ramirez-Pinero, H. Vazquez-Leal, V. M. Jimenez-Fernan dez et al., "Speed-up hyperspheres homotopic path tracking algorithm for PWL circuits simulations," Springerplus, vol. 5, no. 1, pp. 1–29, 2016.

[6] A. Nagy and I. Vajk, "Sequential time optimal path-tracking algorithm for robots," IEEE Transactions on Robotics, vol. 35, no. 5, pp. 1253–1259, 2019.

[7] A. K. Patel and S. Chatterjee, "Computer vision-based lime stone rock-type classification using probabilistic neural network," Geoscience Frontiers, vol. 7, no. 1, pp. 53–60, 2016.

[8] Y. A. Quintero, G. Alvarez, and J. E. Aedo, "Path-tracking algorithm for intelligent transportation systems," IEEE Latin America Transactions, vol. 14, no. 6, pp. 2934–2939, 2016.

[9] E. E. Sánchez-Ramírez, A. J. Rosales-Silva, and R. A. Alfaro Flores, "High-precision visual-tracking using the IMM algo rithm and discrete GPI observers (IMM-DGPIO)," Journal of Intelligent & Robotic Systems, vol. 99, no. 3, pp. 815–835, 2020.

[10] A. A. Konovalov,"Target tracking algorithm for passive coher ent location," IET Radar Sonar & Navigation, vol. 10, no. 7, pp. 1228–1233, 2016.

[11] E. Jaya and B. T. Krishna, "Fuzzy-based MTD: a fuzzy decisive approach for moving target detection in multichannel SAR framework," Data Technologies and Applications, vol. 54, no. 1, pp. 66–84, 2020.

[12] J. Wldchen and P. Mder, "Plant species identification using computer vision techniques: a systematic literature review," Archives of Computational Methods in Engineering, vol. 25, no. 2, pp. 507–543, 2018