Jurnal Elektronik Ilmu Komputer Udayana p-ISSN: 2301-5373
Volume 12, No 2. November 2023 e-ISSN: 2654-5101

Load Time Optimization on React Website using Incremental Static Regeneration with NextJS

Gede Sudimahendra[a1], Luh Arida Ayu Rahning Putri[a2]

[a]Informatika, Universitas Udayana
Jimbaran, Badung, Bali, Indonesia
[1]gedesudimahendra@gmail.com
[2]rahningputri@unud.ac.id

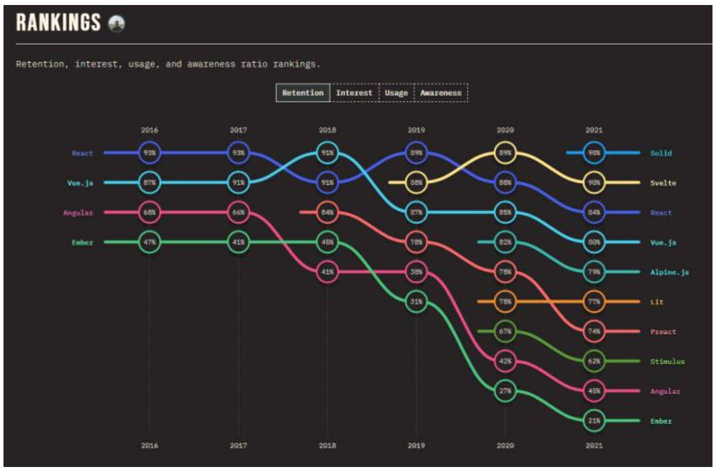
Abstract

There's a lot of tools that can be used to build or develop a website. Starting from basic HTML CSS and JavaScript to the use of UI Framework such as React, Angular, Vue JS or Svelte But, the use of UI Framework doesn't come with no cons. UI Framework like React, use virtual DOM, instead of modifying the DOM directly, so when the first time application load, the framework needs to load library to modify the virtual DOM, before the page can load. This can leads to slow first loading time. This paper research performance improvement when using ISR (Incremental Static Regeneration) in NextJS

Keywords: *User Experience, Front End Development, React, Server Side Rendering*

1. Introduction

There's a lot of tools that can be used to build or develop a website. Starting from basic HTML CSS and JavaScript to the use of UI Framework such as React, Angular, Vue JS or Svelte. Even though the use of HTML CSS and JavaScript is still viable, the lack of good Developer Experience when used building a large-scale application with a lot of cogs and wheels working and a lot of interactivity contributes a lot to the popularity of UI Framework. This can be seen on the last State of JavaScript Survey, where Solid, Svelte, React, and Vue still are the most popular and most used framework.



Picture 1. Framework Popularity [1]

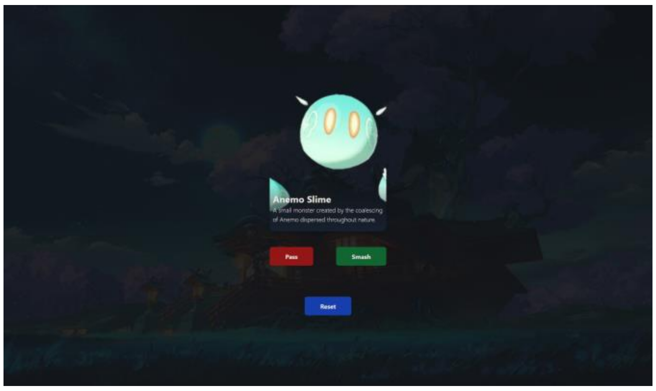
But, the use of UI Framework doesn't come with no cons. UI Framework like React, use virtual DOM, instead of modifying the DOM directly, so when the first time application load, the framework needs to load library to modify the virtual DOM, before the page can load. This can leads to slow first loading time.

This problem created the needs of faster way to load web application initially. Every UI framework has their own SSR solution as a way to combat this problem. This research wants to introduce the SSR solution and also test the SSR solution is that solution really give performance improvement in load-time

2. Reseach Methods

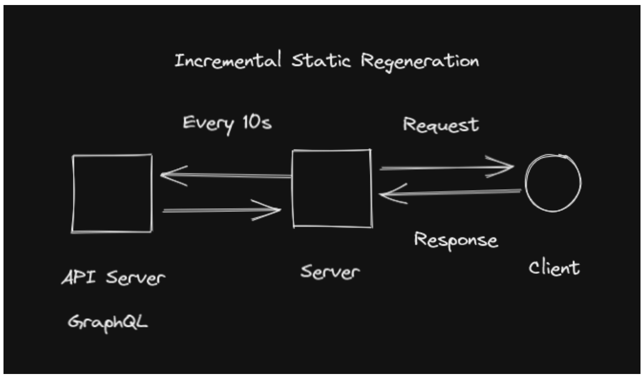
For the UI Framework I use React based on React's popularity based on the last State of JavaScript survey. On the application side, I used application with the following criteria:

1. Data Fetching (Fetch data from external resource)
2. Contains Image, Text, and Basic CSS Styling
3. Contains JavaScript for interactivity


Anemo Slime
A small monster created by the coalescing of Anemo dispersed throughout nature.
Pass
Smash
Reset

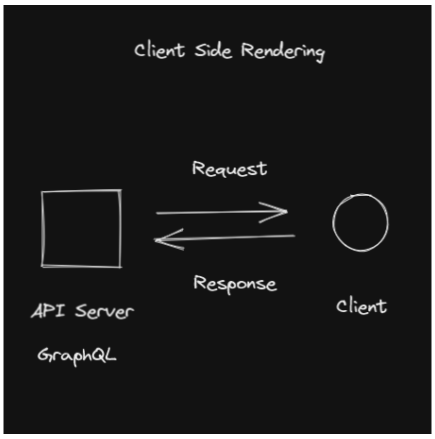
Picture 2. Application Screenshot

This is the application that is used. Genshin Impact Monster Smash and Pass. This application fetch data from external resource from Hygraph's GraphQL. It contains array of 10 object data with name, picture, description and smash and pass number. User can click or choose smash or pass after that it will generate next random monster to choose. The Application is simple enough so that it can be created in a relatively short time and complex enough so that contains all the requirements needed.



Picture 3. Incremental Static Regeneration

Incremental Static Regeneration is the same as Static Site Generation, but the generated site is refreshed after some time. In this case every 10 second. So the client will receive static generated site. This can improve the initial data load time.



Picture 4. Client Side Rendering

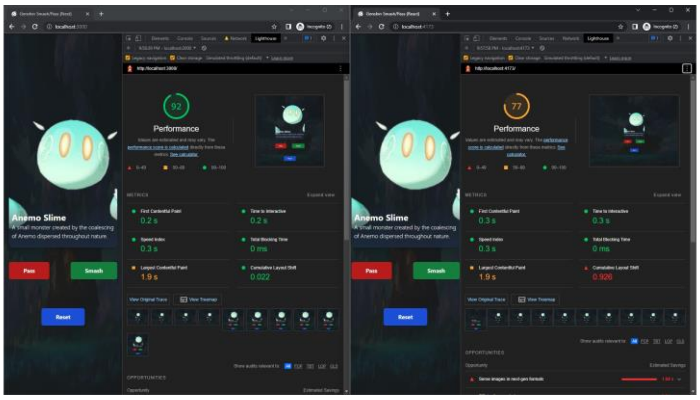
Client Side Rendering request data directly from the API Server and create the view in client side.

For testing method, I use Lighthouse testing. Lighthouse is an open-source tool, pro-viding audits for performance, as well as for accessibility, search engine optimization, and progressive web apps, with indicators on how to improve these aspects of web-sites if needed [2]. Lighthouse is used to get the following value:

1. FCP (First Contentful Paint) : marks the time at which the first text or image is painted [3]
2. LCP (Largest Contentful Paint) : marks the time at which the largest text or image is painted [3]
3. FMP (First Meangingful Paint) : The time when the browser paints the content that users are interested in [3]

Testing will be run 5 times and the result will be average of the 5 test.

3. Result and Discussion



http://localhost:3000/

92
Performance

METRICS

First Contentful Paint 0.2 s
Time to Interactive 0.2 s
Speed Index 0.3 s
Total Blocking Time 0 ms
Largest Contentful Paint 1.9 s
Cumulative Layout Shift 0.022

http://localhost:4173/

77
Performance

METRICS

First Contentful Paint 0.3 s
Time to Interactive 0.3 s
Speed Index 0.3 s
Total Blocking Time 0 ms
Largest Contentful Paint 1.9 s
Cumulative Layout Shift 0.926

Anemo Slime
A small monster created by the coalescing of Anemo dispersed throughout nature.
Pass
Smash
Reset
View Original Trace
View Treemap
OPPORTUNITIES
Opportunity
Estimated Savings
Serve images in next-gen formats

Picture 5. Lighthouse Testing

The lighthouse test that is used is the one native in chrome. Left is Next JS and Right is React. From the test we got 92 Performance Score for NextJS and 72 Performance Score in React. From the test also we can see FCP differs faster by 0.2 seconds for NextJS.

After the test is run 5 times. The result of the test is averaged. And the following is the result of the Lighthouse testing.

Table 1. Lighthouse Test Result

Framework	First Contentful Paint	Largest Contentful Paint	First Meaningful Paint
React	272	1493	272
NextJS	206	1664	206

We can directly see that improvement is quite large in FCP and FMP, but we can see performance decrease in LCP. The improvement in initial load time is quite large.

4. Conclusion

Test result concluded that there’s 24,5% improvement in First Contentful Paint and First Meaningful Paint between React and NextJS. After that we can see decrease speed in LCP. Improvement is not overwhelmingly large. But when paired with a larger scale application, we can look forward to a larger improvement in load time compared to CSR. On the other hand, we need to look

References

[1] Benitter, Rafael, "State of JavaScript 2021”, 2021. [Online]. Available: https://stateofjs.com/en-us/. [04 October 2022]

[2] Hericko, Tjasa, Sumak, Bostjam, Brdnik, Sasa " Towards Representative Web Performance Measurements with Google Lighthouse " *Proceedings of the 2021 7th Student Computer Science Research Conference (StuCoSReC*) 2021.

[3] Google, 2021. [Online]. Available: https://web.dev [04 October 2022]