ISSN: 2598-8565 (media cetak)
ISSN: 2620-4339 (media online)

# IMPLEMENTASI DYNAMIC RENDERING UNTUK OPTIMASI EFISIENSI SERVER DAN INDEKSABILITAS PADA SINGLE PAGE APPLICATION

**Niko Achmad✉, Moh. Ahsan, Ainia Walidaroyani**
Teknik Informatika, Universitas PGRI Kanjuruhan, Malang, Indonesia
Email: nicoachmad15@gmail.com

DOI: https://doi.org/10.46880/jmika.Vol10No1.pp25-34

***ABSTRACT***

*The development of a Client-Side Rendering (CSR)-based Single Page Application (SPA) has search engine indexability weaknesses, while Server-Side Rendering (SSR) overloads server computation. This study aims to prove the efficiency of the Dynamic Rendering architecture as a middle-ground solution. Through Nginx configuration, the server detects the User-Agent to serve static pages (CSR) to human users and fully rendered pages (SSR) to bot crawlers. Experimental testing was conducted on a Virtual Private Server (VPS) using k6 with a constant load of 100 Virtual Users, Prometheus, and Grafana, as well as Document Object Model (DOM) validation via Google Search Console. The Mann-Whitney U Test results proved a significant performance difference with a p-value < 0.05. The implementation of Dynamic Rendering is highly efficient, capable of reducing CPU Utilization by 92.28% and Memory Usage by 5.14% and increasing Request Per Second (RPS) capacity by 17.19%. Indexability validation also confirmed that crawlers successfully received the HTML document entirely. In conclusion, Dynamic Rendering is proven to be an effective architectural solution to minimize server load while ensuring optimal content visibility on search engines.*

***Keyword: Dynamic Rendering, Indexability, Load Testing, Server Efficiency, Single Page Application.***

**ABSTRAK**

*Pengembangan Single Page Application (SPA) berbasis Client Side Rendering (CSR) memiliki kelemahan indeksabilitas mesin pencari, sedangkan Server Side Rendering (SSR) membebani komputasi server. Penelitian ini bertujuan membuktikan efisiensi arsitektur Dynamic Rendering sebagai solusi penengah. Melalui konfigurasi Nginx, server mendeteksi User-Agent untuk menyajikan halaman statis (CSR) kepada pengguna manusia dan halaman hasil render penuh (SSR) kepada bot crawler. Pengujian eksperimental dilakukan pada Virtual Private Server (VPS) menggunakan k6 dengan beban konstan 100 Virtual Users, Prometheus, dan Grafana, serta validasi Document Object Model (DOM) via Google Search Console. Hasil uji Mann-Whitney U Test membuktikan bahwa terdapat perbedaan kinerja yang signifikan dengan p-value < 0,05. Penerapan Dynamic Rendering sangat efisien, mampu menurunkan CPU Utilization sebesar 92,28% dan Memory Usage 5,14%, serta meningkatkan kapasitas Request Per Second (RPS) sebesar 17,19%. Validasi indeksabilitas juga mengonfirmasi crawler berhasil menerima dokumen HTML secara utuh. Sebagai kesimpulan, Dynamic Rendering terbukti menjadi solusi arsitektural yang efektif untuk meminimalkan beban server sekaligus menjamin visibilitas konten secara optimal di mesin pencari.*

***Kata Kunci: Dynamic Rendering, Efisiensi Server, Indeksabilitas, Load Testing, Single Page Application.***

## PENDAHULUAN

Pengembangan aplikasi web modern saat ini didominasi oleh arsitektur *Single Page Application* (SPA) yang menawarkan interaktivitas tinggi bagi pengguna. Namun, SPA dengan metode rendering bawaan yaitu *Client Side Rendering* (CSR) memiliki masalah krusial terkait visibilitas di mesin pencari. Penelitian empiris menunjukkan bahwa *crawler* seringkali gagal mengeksekusi JavaScript secara tepat waktu untuk melihat konten, sehingga aplikasi berisiko tidak terrines dengan baik (Kowalczyk & Szandala, 2024; Talluri, 2023). Sebagai solusi, mekanisme *Server Side Rendering* (SSR) sering diadopsi karena memungkinkan halaman dirender penuh di server sebelum dikirim ke klien, sehingga konten siap diindeks. Akan tetapi, SSR memberikan beban komputasi yang berat karena setiap permintaan diproses ulang secara penuh di server (Hanafi et al., 2024). Hal ini dapat menyebabkan *bottleneck* pada penggunaan CPU dan memori, serta membatasi skalabilitas aplikasi.

Pendekatan *Dynamic Rendering* muncul sebagai solusi arsitektural yang menggabungkan keunggulan CSR dan SSR. Melalui deteksi *User-Agent*,

ISSN: 2598-8565 (media cetak)
ISSN: 2620-4339 (media online)

server menyajikan versi CSR yang ringan untuk pengguna manusia, dan menyajikan versi SSR yang telah direnderer penuh khusus untuk *crawler* mesin pencari. Meskipun pergeseran ke arah arsitektur hibrida ini telah direkomendasikan (Vepsäläinen, 2025), penelitian kuantitatif mengenai seberapa besar efisiensi penghematan sumber daya server yang dihasilkan masih sangat terbatas.

Oleh karena itu, penelitian ini bertujuan untuk menganalisis perbandingan beban komputasi server yang meliputi CPU *Utilization*, Memory *Usage*, dan *Request Per Second* (RPS) antara pendekatan *Server Side Rendering* (SSR) penuh dan *Dynamic Rendering*. Selain itu, penelitian ini juga bertujuan untuk mengevaluasi tingkat efisiensi penggunaan sumber daya server yang dihasilkan oleh penerapan *Dynamic Rendering*, serta memvalidasi kemampuannya dalam mempertahankan indeksabilitas konten pada mesin pencari melalui analisis struktur Document Object Model (DOM).

Kontribusi utama dari penelitian ini adalah memberikan bukti empiris kuantitatif bahwa *Dynamic Rendering* mampu menjadi solusi optimal dalam menyeimbangkan efisiensi server dan kebutuhan Search Engine Optimization (SEO) pada arsitektur Single Page Application. Berdasarkan tujuan tersebut, hipotesis yang diajukan (H1) adalah terdapat perbedaan kinerja yang signifikan antara pendekatan SSR penuh dan *Dynamic Rendering*, di mana *Dynamic Rendering* memberikan efisiensi sumber daya server yang lebih tinggi tanpa mengorbankan indeksabilitas konten.

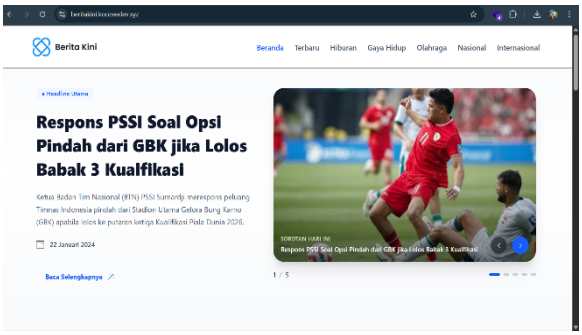
## TINJAUAN PUSTAKA

Penelitian terkait strategi *rendering* web mengonfirmasi bahwa tidak ada satu pun paradigma *rendering* yang optimal secara universal untuk semua kebutuhan. menyimpulkan bahwa platform berskala besar perlu mengadopsi arsitektur hibrida untuk menyeimbangkan antara performa *client-side* dan kebutuhan *Search Engine Optimization* (SEO) (Gangishetti & Jain, 2026). Penggunaan *framework* hibrida juga telah dibuktikan secara empiris oleh Pati & Zaki (2025) mampu meningkatkan performa kecepatan muat serta memberikan skor SEO yang lebih unggul dibandingkan aplikasi yang murni berbasis CSR (Pati & Zaki, 2025). Hal ini sejalan dengan temuan Talluri (2023) yang menegaskan bahwa implementasi *Server-Side Rendering* atau *Dynamic Rendering* sangat esensial untuk mengatasi keterbatasan indeksabilitas pada arsitektur dasar SPA (Talluri, 2023).

Dilema utama terletak pada beban infrastruktur. Evaluasi kinerja oleh penelitian terdahulu menemukan bahwa meskipun metode SSR memberikan keunggulan SEO, pendekatan ini memberikan beban yang berat karena seluruh proses dibebankan pada server (Alvian Noer & Suartana, 2024; Fadhilah Iskandar et al., 2020; Hanafi et al., 2024; Pramana Putra & Puspita Sari, 2024; Rahman Hidayatullah & Suartana, 2025; Vallamsetla, 2024). Hal ini memicu kebutuhan akan metode *Dynamic Rendering* yang secara cerdas memanfaatkan peran *Reverse Proxy*, seperti Nginx, sebagai pengatur lalu lintas (*traffic controller*). Nginx dapat dikonfigurasi menggunakan logika *conditional routing* untuk mendeteksi identitas *HTTP User-Agent*. Permintaan dari *bot* secara otomatis diarahkan ke layanan SSR, sedangkan pengguna biasa dilayani dengan berkas statis, sehingga menekan konsumsi metrik kinerja kritis seperti utilisasi CPU dan penggunaan memori. Dari kajian empiris ini, hipotesis penelitian dikembangkan bahwa pemisahan jalur *rendering* berdasarkan *User-Agent* akan secara drastis memangkas beban kerja server tanpa menghilangkan struktur *Document Object Model* (DOM) yang dibutuhkan untuk indeksabilitas.

## METODE PENELITIAN

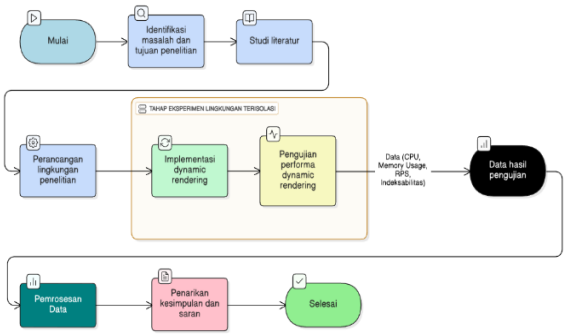
Penelitian ini menggunakan rancangan eksperimental dengan pendekatan kuantitatif yang dilakukan pada lingkungan yang terisolasi. Ruang lingkup objek pengujian adalah sebuah *Single Page Application* (SPA) berupa portal berita yang dibangun menggunakan *framework* Nuxt.js 3 yang terbukti efektif meningkatkan performa aplikasi berbasis web (Angkasa et al., 2023), dengan fokus pengujian pada halaman beranda (*homepage*).




**Gambar 1**. Objek Penelitian Web Portal Berita

Secara sistematis, tahapan penelitian ini diuraikan melalui alur kerja komprehensif yang dapat dilihat pada Gambar 2.

ISSN: 2598-8565 (media cetak)
ISSN: 2620-4339 (media online)




**Gambar 2**. Alur Penelitian

Keterangan:

1. Identifikasi masalah dan tujuan penelitian: Tahap awal untuk mengidentifikasi inefisiensi pada arsitektur *Server Side Rendering* (SSR) yang membebani server dan keterbatasan *Client Side Rendering* (CSR) dalam aspek SEO, dilanjutkan dengan merumuskan tujuan perancangan sistem *Dynamic Rendering* yang efisien.
2. Studi literatur: Pengkajian mendalam terhadap teori dasar dan penelitian terdahulu yang relevan dengan mekanisme kerja *web rendering*, metrik performa server, dan teknis indeksabilitas mesin pencari.
3. Perancangan lingkungan penelitian: Proses merancang topologi infrastruktur pada satu unit *Virtual Private Server* (VPS), dengan memposisikan Nginx sebagai gerbang utama (*gateway*) yang mengatur lalu lintas berdasarkan deteksi *User-Agent*.
4. Tahap Eksperimen Lingkungan Terisolasi: Merupakan fase inti penelitian yang terdiri dari dua sub-tahap utama:

a. Implementasi *dynamic rendering*: Mengonfigurasi aplikasi agar berjalan dalam mode hibrida (SSR dan CSR) dan memprogram Nginx menggunakan logika *conditional routing*.
b. Pengujian performa *dynamic rendering*: Pelaksanaan *load testing* untuk mensimulasikan lalu lintas menggunakan dua skenario komparatif, yaitu skenario kontrol (akses *bot* / SSR penuh) dan skenario eksperimen (akses pengguna manusia / *Dynamic Rendering*).

5. Data hasil pengujian: Tahap pengumpulan *output* dari eksperimen yang mencakup metrik *CPU Utilization*, *Memory Usage*, *Request Per Second* (RPS), dan status indeksabilitas HTML.
6. Pemrosesan Data: Analisis data metrik menggunakan metode deskriptif kuantitatif untuk menghitung persentase tingkat optimasi yang dicapai, serta melakukan validasi fungsional fungsional teknis terhadap struktur HTML (*Document Object Model*) untuk mesin pencari.
7. Penarikan kesimpulan dan saran: Tahap akhir untuk menjawab hipotesis penelitian terkait keberhasilan pencapaian efisiensi sumber daya dan validitas indeksabilitas, serta menyusun saran bagi pengembangan penelitian selanjutnya.

Penelitian ini menggunakan pendekatan eksperimental kuantitatif yang dilakukan pada lingkungan *Virtual Private Server* (VPS) yang terisolasi. Variabel bebas dalam penelitian ini adalah mekanisme *rendering*, yang terbagi menjadi kondisi kontrol (*Server Side Rendering* penuh) dan kondisi eksperimen (*Dynamic Rendering*). Variabel terikat yang diukur mencakup *CPU Utilization*, *Memory Usage*, *Request Per Second* (RPS), dan validitas indeksabilitas konten. Sementara itu, variabel kontrol dijaga konstan dengan menggunakan spesifikasi server yang seragam selama pengujian berlangsung. Spesifikasi *Virtual Private Server* (VPS) dapat dilihat pada Tabel 1.

**Tabel 1**. Spesifikasi Lingkungan Server

| No. | Parameter | Spesifikasi |
|---|---|---|
| 1. | Sistem Operasi | Ubuntu Server 22.04 LTS (64-bit) |
| 2. | CPU | 1vCPU |
| 3. | RAM | 2 GB RAM |
| 4. | Penyimpanan | 20 GB SSD |
| 5. | *Web Server* | Nginx (*User-Agent routing*) + PM2 |
| 6. | *Framework* | Nuxt.js 3 |

Teknik analisis data dalam penelitian ini menggunakan analisis statistik inferensial untuk data performa server. Untuk data performa komputasi (CPU, Memori, dan RPS), tahapan analisis diawali dengan pembersihan data (*data cleaning*). Data mentah (*raw data*) yang diperoleh dari sistem pemantauan Prometheus dipotong (*trimming*) pada interval awal (*fase ramp-up*) dan akhir pengujian (*fase ramp-down*). Langkah ini krusial untuk mengeliminasi varians data yang ekstrem akibat inisialisasi atau pemutusan koneksi massal, sehingga perbandingan performa murni difokuskan pada kondisi beban stabil (*fase steady state*).

Data yang telah dibersihkan kemudian melalui uji prasyarat normalitas menggunakan metode *Shapiro-Wilk* (Agustin & Permatasari, 2020). Metode ini dipilih karena jumlah sampel data yang dihasilkan tergolong kecil ($n < 50$). Berdasarkan karakteristik metrik performa server yang sangat fluktuatif (*spikes*) dan

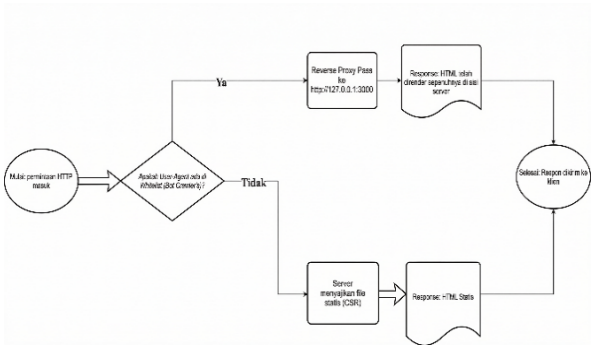
umumnya tidak berdistribusi normal (nilai $p < 0,05$), pengujian hipotesis untuk membandingkan kedua skenario dilakukan menggunakan uji beda non-parametrik *Mann-Whitney U Test* dengan taraf signifikansi 5% ($\alpha = 0,05$).

Setelah signifikansi perbedaan dibuktikan, analisis dilanjutkan dengan menghitung tingkat efisiensi secara matematis menggunakan nilai rata-rata (*mean*) dari masing-masing skenario. Perhitungan efisiensi beban kerja komputasi (CPU dan Memori) menggunakan rumus persentase penurunan relatif, sedangkan kapasitas layanan (RPS) menggunakan rumus persentase peningkatan relatif.

**Konfigurasi Lingkungan Penelitian**

Tahap konfigurasi lingkungan penelitian difokuskan pada penerapan manipulasi variabel bebas, yaitu konfigurasi mekanisme *rendering* pada server untuk merespons dua kondisi perlakuan tanpa mengubah fitur pengembangan aplikasi. Aplikasi berbasis Nuxt.js disiapkan untuk beroperasi dalam dua mode secara simultan. Pada mode statis (CSR), aplikasi hasil kompilasi ditempatkan pada direktori publik di /var/www/appcsr untuk melayani permintaan dari pengguna manusia. Pada mode server (SSR), aplikasi yang sama dijalankan menggunakan *process manager* (PM2) pada *port* lokal 3000 untuk melayani *bot crawler* mesin pencari.

Web server Nginx kemudian dikonfigurasi sebagai gerbang logika (*logic gateway*) berkinerja tinggi yang menerapkan *conditional routing*. Nginx diprogram untuk mendeteksi *HTTP Header User-Agent* dan mencocokkannya dengan daftar putih (*whitelist*) yang memuat *string* identifikasi bot seperti Googlebot, Bingbot, dan Yandex. Pada Gambar 3, jika *User-Agent* cocok dengan daftar tersebut, lalu lintas diteruskan (*proxy_pass*) ke layanan SSR, sedangkan jika tidak ada kecocokan, Nginx akan menyajikan berkas statis CSR dari direktori publik.




**Gambar 3**. Alur *Conditional Routing* Nginx

**Prosedur Pengumpulan Data**

Prosedur pengumpulan data dirancang dalam dua skema utama untuk memastikan data bersifat valid, reliabel, dan bebas bias, yaitu pengumpulan data performa sisi server dan validasi indeksabilitas. Pertama, pengumpulan data performa bertujuan mengukur CPU *Utilization*, *Memory Usage*, dan *Request Per Second* (RPS) menggunakan instrumen load testing k6 serta sistem pemantauan Prometheus dengan visualisasi Grafana (Jani, 2024; Putra et al., 2024). Pengujian ini menggunakan skenario beban konstan (*constant load*) sebesar 100 *Virtual Users* (VUs) yang berjalan paralel selama 5 menit untuk mendapatkan nilai rata-rata kinerja. Protokol *Sequential Testing* diterapkan dengan jeda waktu pendinginan (*cooldown*) selama 10 menit antar pengujian agar resource server kembali ke kondisi nol (*idle state*). Pengujian dibagi menjadi:

1. Skenario A (Simulasi Beban SSR): Skrip k6 dikonfigurasi untuk mengirimkan header *User-Agent*: Googlebot/2.1. Lalu lintas ini secara otomatis diteruskan oleh Nginx ke layanan Node.js, merepresentasikan kondisi performa dasar sebelum optimasi.
2. Skenario B (Simulasi Beban *Dynamic*): Skrip k6 dikonfigurasi untuk mengirimkan header *User-Agent*: Mozilla/5.0, Chrome/91.0. Permintaan ini tidak masuk dalam whitelist Nginx, sehingga dilayani langsung dengan berkas statis tanpa membebani proses Node.js.

Kedua, prosedur validasi indeksabilitas berfungsi untuk memastikan mekanisme kondisional berjalan sesuai rancangan, yang dilakukan melalui alat URL Inspection pada Google Search Console. Pengujian diawali dengan mengakses halaman melalui peramban web standar dan memeriksa kode sumber (*View Page Source*) untuk memverifikasi bahwa HTML hanya berisi kerangka dasar minimal seperti  tanpa teks utama, yang merupakan indikator penerapan CSR. Setelah itu, pengujian dilanjutkan dengan fitur *Test Live* URL di Google Search Console untuk menyimulasikan pandangan Googlebot. Hasil *View Crawled Page* harus memuat dokumen HTML lengkap beserta seluruh konten utama yang di *render* di sisi server. Adanya kontras antara dokumen kosong di peramban dan dokumen utuh di mesin pencari menjadi bukti empiris yang memvalidasi keberhasilan penerapan *Dynamic Rendering*.

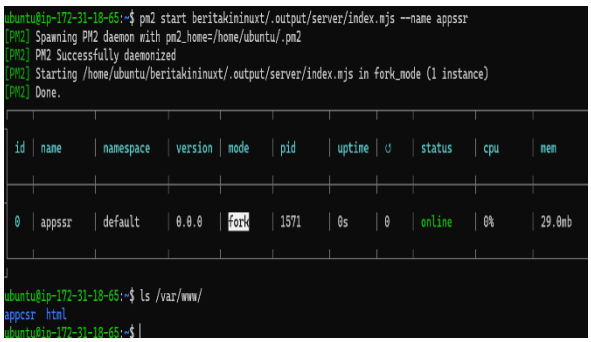
**HASIL DAN PEMBAHASAN**

Berdasarkan perancangan yang telah dilakukan, pengujian dilakukan untuk mengevaluasi kinerja

*Dynamic Rendering* dalam mengoptimasi efisiensi server dan indeksabilitas konten.

**Implementasi Mekanisme Dynamic Rendering**

Berdasarkan perancangan yang telah dilakukan, tahap awal difokuskan pada implementasi konfigurasi web server Nginx yang bertindak sebagai pengatur lalu lintas kondisional (*conditional routing*). Aplikasi SPA berbasis Nuxt.js diimplementasikan dalam dua mode yang berjalan secara simultan pada satu server. Mode statis (CSR) ditempatkan pada direktori publik, sementara mode *Server Side Rendering* (SSR) dijalankan menggunakan *process manager* (PM2) pada *port* lokal 3000. Nginx berhasil dikonfigurasi untuk mendeteksi *HTTP Header User-Agent*; meneruskan permintaan *bot crawler* (seperti Googlebot) ke layanan SSR, dan langsung menyajikan berkas statis CSR kepada pengguna manusia tanpa membebani pemrosesan Node.js.




**Gambar 4**. Implementasi Mekanisme *Dynamic Rendering*

**Hasil Pengujian Kinerja Server**

Evaluasi efisiensi komputasi dilakukan melalui simulasi beban menggunakan instrumen k6 dengan skenario beban konstan sebesar 100 *Virtual Users* selama 5 menit. Pengujian membandingkan Skenario A (sistem berjalan pada mode SSR penuh) dan Skenario B (sistem menerapkan optimasi *Dynamic Rendering*). Rekapitulasi perbandingan nilai rata-rata (*mean*) dari metrik kinerja server setelah melewati fase pembersihan data disajikan pada Tabel 2.

**Tabel 2**. Rata-rata Hasil Pengujian Kinerja Server

| Metrik Kinerja | Skenario A | Skenario B |
|---|---|---|
| *CPU Utilization* (%) | 25,43 | 1,96 |
| *Memory Usage* (%) | 26,30 | 24,94 |
| *Request Per Second*(req/s) | 51,46 | 62,14 |

Tabel 2 menunjukkan bahwa beban *CPU Utilization* pada metode SSR penuh mencapai 25,43% akibat proses *rendering* halaman HTML yang dilakukan berulang kali di sisi server. Penerapan *Dynamic Rendering* berhasil menekan angka tersebut secara drastis menjadi 1,96% karena permintaan mayoritas (dari simulasi peramban manusia) dilayani langsung dengan respons berkas statis. Kapasitas *throughput* sistem (*Request Per Second*) juga mengalami peningkatan dari 51,46 menjadi 62,14 permintaan per detik, yang menandakan server mampu melayani lebih banyak pengguna dalam satu waktu akibat berkurangnya beban pemrosesan. Penurunan CPU *Utilization* yang sangat signifikan menunjukkan bahwa proses rendering yang sebelumnya menjadi *bottleneck* utama pada SSR berhasil diminimalkan. Hal ini mengindikasikan bahwa *Dynamic Rendering* secara efektif mengalihkan sebagian besar beban komputasi dari server ke sisi klien untuk permintaan pengguna manusia.

Selain itu, peningkatan nilai *Request Per Second* (RPS) menunjukkan adanya peningkatan throughput sistem. Dengan berkurangnya beban pemrosesan di sisi server, sistem mampu melayani lebih banyak permintaan dalam waktu yang sama, yang menjadi indikator penting dalam peningkatan performa sistem secara keseluruhan.

Sementara itu, penurunan Memory Usage yang relatif kecil menunjukkan bahwa optimasi utama dari Dynamic Rendering lebih berpengaruh pada CPU dibandingkan memori, yang sejalan dengan karakteristik proses rendering yang bersifat CPU-intensive.

**Analisis Uji Hipotesis dan Tingkat Efisiensi**

Seluruh data hasil pengujian pada fase stabil (*steady state*) dianalisis menggunakan uji prasyarat normalitas *Shapiro-Wilk* yang dapat dilihat di Tabel 3.

**Tabel 3**. Hasil Uji Normalitas

| Metrik Kinerja | Sig. *Value* | |
|---|---|---|
| | Skenario A | Skenario B |
| *CPU Utilization* | .001 | .001 |
| *Memory Usage* | .000 | .000 |
| *Request Per Second* | .000 | .000 |

Karena data tidak berdistribusi normal nilai signifikansi (*p-value*) lebih kecil dari 0,05, evaluasi dilanjutkan menggunakan uji non-parametrik *Mann-Whitney U Test*. Hasil uji statistik menunjukkan nilai probabilitas *Asymp. Sig. (2-tailed)* kurang dari 0,05 untuk ketiga metrik yang diuji (CPU, Memori, dan RPS). Hal ini membuktikan bahwa Hipotesis Alternatif (H1) diterima, yang berarti terdapat perbedaan kinerja yang signifikan secara statistik antara metode SSR penuh dan *Dynamic Rendering* yang dapat dilihat di Tabel 4.

ISSN: 2598-8565 (media cetak)
ISSN: 2620-4339 (media online)

**Tabel 4**. Rekapitulasi Hasil Uji Hipotesis

| Metrik Kinerja | Asymp. Sig. (2-tailed) | Hasil Hipotesis |
|---|---|---|
| *CPU Utilization* | < 0,001 | *Ho* ditolak |
| *Memory Usage* | < 0,001 | *Ho* ditolak |
| *Request Per Second* | 0,013 | *Ho* ditolak |

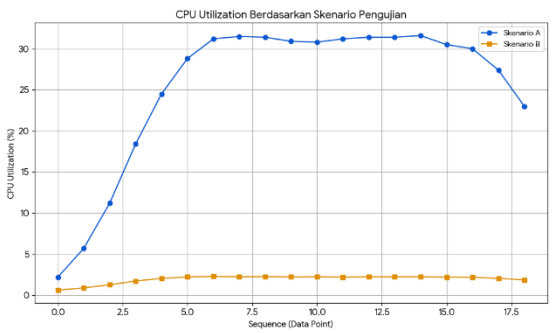
Secara matematis, tingkat efisiensi yang dihasilkan dari penerapan *Dynamic Rendering* sangat optimal yang dapat dihitung melalui rumus-rumus dibawah ini.

$$\textbf{Efisiensi CPU} = \left(\frac{25,43 - 1,96}{25,43}\right) X\ 100\% = 92,28\%$$

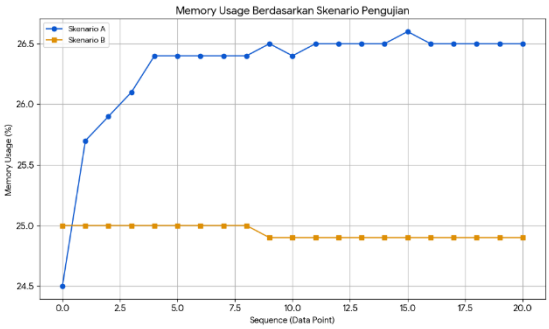
$$\textbf{Efisiensi Memori} = \left(\frac{26,29 - 24,94}{26,29}\right) X\ 100\% = 5,14\%$$

$$\textbf{Peningkatan RPS} = \left(\frac{62,14 - 51,46}{62,14}\right) X\ 100\% = 17,19\%$$

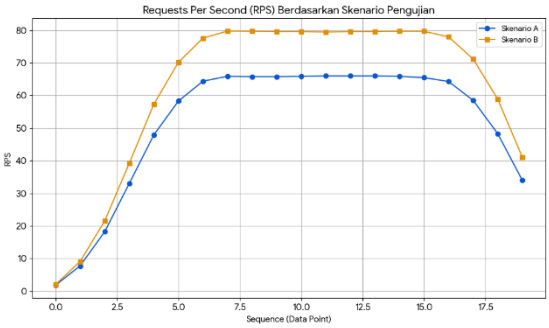
Berdasarkan hasil perhitungan pada masing-masing rumus, membuktikkan bahwa arsitektur *Dynamic Rendering* mampu menurunkan *CPU Utilization* sebesar 92,28% dan *Memory Usage* sebesar 5,14%. Pada saat yang sama, kapasitas sistem untuk memproses permintaan (*Request Per Second*) meningkat sebesar 17,19% dibandingkan saat menggunakan arsitektur SSR penuh. Grafik hasil perbedaan hasil pengujian dapat dilihat pada Gambar 5, Gambar 6, dan Gambar 7.




**Gambar 5**. Grafik CPU *Utilization*




**Gambar 6**. Grafik *Memory Usage*

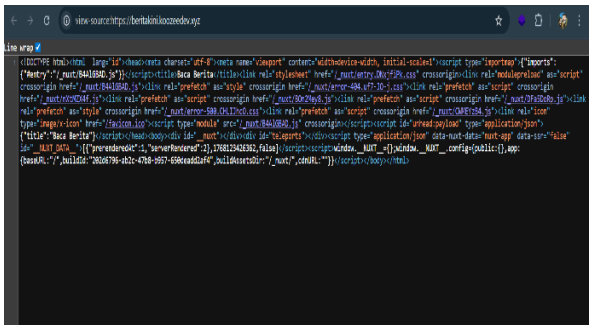



**Gambar 7**. Grafik *Request Per Second*

**Validasi Fungsional Indeksabilitas**

Validasi indeksabilitas dilakukan untuk memastikan bahwa penghematan sumber daya komputasi di sisi server tidak berdampak negatif terhadap visibilitas konten pada *Search Engine Optimization* (SEO). Pengujian dilakukan secara fungsional melalui komparasi inspeksi struktur *Document Object Model* (DOM) yang diterima oleh dua jenis *User-Agent* yang berbeda. Pada pengujian pertama, halaman web diakses menggunakan peramban standar (seperti Google Chrome) yang mewakili lalu lintas pengguna biasa. Pemeriksaan dilakukan melalui fitur *View Page Source*. Hasil inspeksi menunjukkan bahwa kode sumber HTML yang diunduh dari server hanya berupa kerangka dasar aplikasi berserta elemen penampung kosong, yaitu . Tidak ada teks konten utama maupun elemen metadata spesifik yang dirender di dalam HTML awal tersebut. Hal ini membuktikan bahwa Nginx berhasil mengenali *User-Agent* manusia dan langsung menyajikan berkas statis (*Client Side Rendering*), di mana proses *rendering* konten selanjutnya diserahkan sepenuhnya kepada mesin peramban klien.

ISSN: 2598-8565 (media cetak)
ISSN: 2620-4339 (media online)



**Gambar 8**. Inspeksi Struktur HTML pada Akses Browser Pengguna

Pada pengujian kedua, simulasi akses mesin pencari dilakukan menggunakan fitur *Test Live URL* pada Google Search Console yang bertindak sebagai Googlebot. Hasil inspeksi pada tab *View Crawled Page* menunjukkan struktur HTML yang sangat berbeda dibandingkan akses pengguna. Seluruh elemen penyusun konten, mulai dari *metadata* hingga teks artikel, telah tersusun lengkap di dalam kode sumber HTML sebelum dieksekusi oleh JavaScript.


**Gambar 9**. Inspeksi Struktur HTML pada Google Search Console

Untuk memperjelas perbedaan hasil rendering dari kedua pengujian tersebut, komparasi ketersediaan elemen kunci DOM disajikan pada Tabel 5.

**Tabel 5**. Komparasi Struktur DOM

| Elemen Kunci DOM | Peramban Pengguna | *Bot Crawler* / Googlebot |
|---|---|---|
| Wadah Aplikasi ( ) | Tersedia (Belum Terisi Konten) | Tersedia (Terisi Konten Penuh) |
| Elemen Metadata (<title>, <meta description>) | Tidak Tersedia / Statis Dasar | Tersedia (Sesuai Konten Dinamis) |
| Elemen Konten Semantik (Tag <h1>, <p>) | Tidak Tersedia | Tersedia |

Berdasarkan Tabel 5, terlihat kontras yang sangat jelas bahwa mekanisme *conditional routing* pada *Dynamic Rendering* berfungsi secara sempurna. Elemen-elemen krusial untuk kebutuhan SEO seperti *tag title*, *meta description*, dan *heading* (<h1>) mutlak dibutuhkan oleh *crawler* untuk memahami konteks halaman web. Ketersediaan elemen-elemen tersebut secara utuh pada skenario pengujian *bot crawler* menegaskan bahwa meskipun mayoritas beban pemrosesan server dipangkas (efisiensi CPU turun 92,28%), arsitektur ini tetap menjamin visibilitas situs dan indeksabilitas informasi secara optimal di mesin pencari.

## Implikasi Terhadap Skalabilitas Sistem dan Lingkungan Produksi

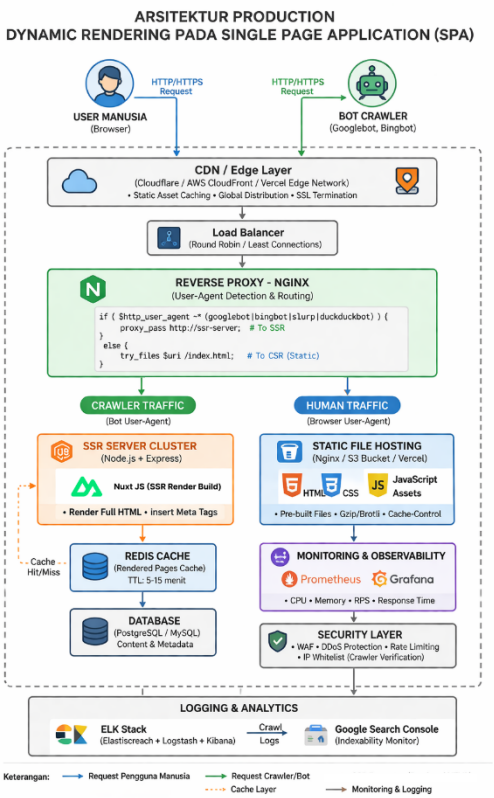
Hasil penelitian menunjukkan bahwa penurunan CPU Utilization sebesar 92,28% memiliki implikasi yang sangat signifikan terhadap skalabilitas sistem. Pada arsitektur *Server Side Rendering* (SSR) penuh, setiap permintaan pengguna memicu proses rendering di sisi server, sehingga beban komputasi meningkat secara linear terhadap jumlah permintaan yang masuk. Kondisi ini berpotensi menyebabkan bottleneck pada sumber daya server, terutama CPU, ketika terjadi lonjakan trafik dalam skala besar.

Sebaliknya, pada pendekatan *Dynamic Rendering*, mayoritas permintaan dari pengguna manusia dilayani melalui berkas statis *Client Side Rendering* (CSR), sehingga proses *rendering* di sisi server hanya terjadi pada permintaan dari bot *crawler*. Hal ini menyebabkan beban komputasi server menjadi jauh lebih ringan dan tidak meningkat secara signifikan meskipun jumlah pengguna bertambah. Dengan demikian, sistem memiliki kemampuan yang lebih baik dalam menangani concurrent users dalam jumlah besar tanpa memerlukan peningkatan spesifikasi server secara signifikan (*vertical scaling*), sehingga lebih efisien dari sisi biaya operasional infrastruktur.

Dalam konteks implementasi pada lingkungan produksi, pendekatan *Dynamic Rendering* juga memberikan fleksibilitas yang tinggi untuk diintegrasikan dengan berbagai teknologi pendukung seperti *Content Delivery Network* (CDN), *reverse*

ISSN: 2598-8565 (media cetak)
ISSN: 2620-4339 (media online)

*proxy caching*, serta *load balancing*. Penggunaan CDN memungkinkan distribusi konten statis kepada pengguna dilakukan dari *edge* server terdekat, sehingga mengurangi latensi dan beban pada server utama. Selain itu, penerapan mekanisme *caching* pada jalur SSR khusus untuk *crawler* dapat menghindari proses rendering ulang terhadap halaman yang sama, sehingga semakin meningkatkan efisiensi sistem secara keseluruhan. Gambar 10 menunjukkan contoh arsitektur implementasi *Dynamic Rendering* pada lingkungan produksi yang mendukung skalabilitas sistem serta optimasi indeksabilitas oleh *crawler* mesin pencari.




**Gambar 10**. Arsitektur Lingkungan Produksi *Dynamic Rendering* pada SPA

Berdasarkan Gambar 10, terlihat bahwa sistem memanfaatkan mekanisme deteksi *User-Agent* pada *reverse proxy* untuk membedakan permintaan antara pengguna dan crawler. Permintaan dari *crawler* diarahkan ke server rendering berbasis Nuxt.js untuk menghasilkan HTML statis yang telah terstruktur dengan baik, sehingga meningkatkan indeksabilitas konten oleh mesin pencari. Hal ini memungkinkan *crawler* seperti Googlebot untuk mengakses konten tanpa harus mengeksekusi JavaScript secara kompleks, yang berdampak langsung pada peningkatan kualitas SEO.

Dari sisi skalabilitas, pemisahan jalur ini memungkinkan sistem untuk mengurangi beban pada server *rendering* karena hanya sebagian kecil trafik (*crawler*) yang diproses di sisi server. Sementara itu, mayoritas pengguna dilayani melalui *static file hosting* yang didukung oleh CDN, sehingga meningkatkan kemampuan sistem dalam menangani lonjakan trafik. Selain itu, penggunaan *caching* seperti Redis pada hasil rendering HTML memungkinkan pengurangan latency dan peningkatan *response time* secara signifikan, yang juga berkontribusi pada performa SEO, khususnya pada metrik *Core Web Vitals*. Dengan demikian, pendekatan *Dynamic Rendering* tidak hanya efisien secara komputasi, tetapi juga memberikan keuntungan strategis dalam implementasi sistem berskala besar di lingkungan produksi.

Dari sisi indeksabilitas konten, hasil validasi menunjukkan bahwa *Dynamic Rendering* mampu menyajikan struktur HTML yang lengkap kepada bot crawler, termasuk elemen-elemen penting seperti tag **<title>, <meta description>,** serta struktur konten semantik seperti **<h1>** dan **<p>**. Ketersediaan elemen-elemen ini sangat krusial dalam proses *crawling* dan *indexing*, karena crawler mesin pencari bergantung pada struktur *Document Object Model* (DOM) untuk memahami konteks dan relevansi suatu halaman web. Dengan disajikannya konten yang telah dirender secara penuh (SSR) kepada *crawler*, maka risiko kegagalan eksekusi JavaScript yang sering terjadi pada pendekatan CSR dapat dihindari.

Implikasi lebih lanjut terhadap *Search Engine Optimization* (SEO) adalah meningkatnya visibilitas konten pada mesin pencari. Dengan struktur HTML yang lengkap dan dapat diakses secara langsung oleh *crawler*, halaman web memiliki peluang yang lebih besar untuk terindeks secara optimal dan muncul pada hasil pencarian yang relevan. Selain itu, penyajian konten yang konsisten antara yang diterima crawler dan pengguna juga membantu menghindari potensi penalti dari mesin pencari akibat praktik *cloaking* yang tidak sesuai pedoman.

Meskipun demikian, penerapan *Dynamic Rendering* pada lingkungan produksi juga memiliki tantangan yang perlu diperhatikan. Salah satunya adalah potensi penyalahgunaan *User-Agent* (*User-Agent spoofing*), di mana permintaan dari pengguna biasa dapat menyamar sebagai bot untuk memperoleh konten SSR. Oleh karena itu, diperlukan mekanisme validasi tambahan seperti verifikasi IP *crawler* (*reverse DNS lookup*) untuk memastikan keaslian identitas bot. Selain itu, kompleksitas konfigurasi pada *reverse proxy* seperti Nginx juga menuntut pengelolaan yang lebih cermat agar sistem tetap stabil dan aman.

Secara keseluruhan, hasil penelitian ini menunjukkan bahwa *Dynamic Rendering* tidak hanya memberikan efisiensi signifikan pada sisi server, tetapi juga mampu meningkatkan skalabilitas sistem serta menjaga kualitas indeksabilitas konten yang berdampak langsung pada performa SEO. Dengan pendekatan yang tepat, arsitektur ini dapat menjadi solusi yang efektif dan praktis untuk diterapkan pada aplikasi berbasis Single Page Application (SPA) dalam skala produksi.

## KESIMPULAN

Berdasarkan hasil pengujian eksperimental dan analisis data yang telah dilakukan, dapat disimpulkan bahwa penerapan arsitektur *Dynamic Rendering* merupakan solusi yang sangat efektif untuk mengoptimalkan efisiensi komputasi server pada *Single Page Application* (SPA) tanpa mengorbankan indeksabilitas mesin pencari. Hasil pengujian hipotesis membuktikan adanya perbedaan kinerja yang signifikan secara statistik antara mekanisme *Server Side Rendering* (SSR) penuh dan *Dynamic Rendering*. Penerapan *Dynamic Rendering* berhasil memberikan efisiensi yang masif dengan menurunkan *CPU Utilization* sebesar 92,28% dan *Memory Usage* sebesar 5,14%, serta meningkatkan kapasitas layanan (*Request Per Second*) sebesar 17,19%.

Selain efisiensi infrastruktur, validasi teknis melalui Google Search Console mengonfirmasi bahwa mekanisme *conditional routing* pada Nginx beroperasi dengan akurat. Server secara cerdas menyajikan halaman berbasis statis yang ringan (*Client Side Rendering*) kepada peramban pengguna, dan secara bersamaan menyajikan struktur dokumen HTML yang utuh (*Server Side Rendering*) kepada *bot crawler* (seperti Googlebot). Dengan demikian, *Dynamic Rendering* terbukti berhasil menjadi arsitektur penengah yang ideal; meminimalkan beban infrastruktur sekaligus memastikan konten web tetap optimal untuk *Search Engine Optimization* (SEO).

## SARAN

1. Penguatan Keamanan *Routing*: Mengingat penelitian ini menggunakan deteksi *User-Agent* standar, disarankan untuk menambahkan mekanisme validasi IP *crawler* (seperti *reverse DNS lookup*) guna memitigasi risiko *User-Agent Spoofing* atau serangan DDoS pada *layer* aplikasi.
2. Integrasi *Caching* Tingkat Lanjut: Untuk lebih menekan beban pada layanan SSR yang melayani *bot*, penelitian selanjutnya dapat mengimplementasikan metode *caching* (misalnya Redis atau *HTTP Cache Headers*) agar server tidak perlu merender ulang halaman yang sama dalam waktu berdekatan.
3. Pengukuran Metrik Sisi Pengguna: Penelitian ini berfokus pada efisiensi infrastruktur server. Studi berikutnya dapat diperluas dengan mengukur metrik *Core Web Vitals* seperti *Largest Contentful Paint* (LCP) dan *Cumulative Layout Shift* (CLS) untuk mengevaluasi dampaknya secara langsung terhadap pengalaman pengguna.
4. Pengujian Lingkungan Terdistribusi: Mengingat arsitektur ini diuji pada *vertical environment* (satu unit VPS), disarankan untuk menguji efektivitas *Dynamic Rendering* dalam skala yang lebih masif menggunakan lingkungan terdistribusi seperti *Load Balancer* atau Kubernetes.

## DAFTAR PUSTAKA

Agustin, P., & Permatasari, R. (2020). Pengaruh Pendidikan dan Kompensasi Terhadap Kinerja Divisi Newproduct Development (Npd) Pada PT. Mayora Indah Tbk. *Jurnal Ilmiah M-Progress, 10*(2). https://doi.org/10.35968/m-pu.v10i2.442

Alvian, N. M., & Suartana, I. M. (2024). Perbandingan Mekanisme Rendering Untuk Optimasi Website Dengan Studi Kasus Website Penitipan Hewan. *Journal of Informatics and Computer Science*, *06*. https://doi.org/10.26740/jinacs.v6n02.p522-531

Angkasa, H., Farell, D., Wijaya, E., Achmad, S., & Fitrianah, D. (2023). *Improving Universal Rendering Performance on NuxtJS-based Web Application*. https://doi.org/10.1109/CITSM60085.2023.10455297

Fadhilah, I. T., Lubis, M., Fabrianti Kusumasari, T., & Ridho Lubis, A. (2020). Comparison between client-side and server-side rendering in the web development. *IOP Conference Series: Materials Science and Engineering*, *801*(1). https://doi.org/10.1088/1757-899X/801/1/012136

Gangishetti, S., & Jain, V. (2026). Comparative Analysis of Client-Side vs. Server-Side Rendering for Large-Scale Content Platforms. *International Journal of AI, BigData, Computational and Management Studies*, *7*(1), 74–80. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V7I1P112

Hanafi, R., Haq, A., & Agustin, N. (2024). Comparison of Web Page Rendering Methods Based on Next.js Framework Using Page Loading Time Test. *Teknika*, *13*(1), 102–108. https://doi.org/10.34148/teknika.v13i1.769

Jani, Y. (2024). Unified Monitoring for Microservices: Implementing Prometheus and Grafana for Scalable Solutions. *Journal of Artificial Intelligence, Machine Learning and*

ISSN: 2598-8565 (media cetak)
ISSN: 2620-4339 (media online)

*Data Science*, *2*(1), 848–852. https://doi.org/10.51219/jaimld/yash-jani/206

Kowalczyk, K., & Szandala, T. (2024). Enhancing SEO in Single-Page Web Applications in Contrast With Multi-Page Applications. *IEEE Access*, *12*, 11597–11614. https://doi.org/10.1109/ACCESS.2024.3355740

Pati, S., & Zaki, Y. (2025). *Evaluating the Efficacy of Next.js: A Comparative Analysis with React.js on Performance, SEO, and Global Network Equity*. https://doi.org/10.48550/arXiv.2502.15707

Pramana, P. H., & Puspita Sari, A. (2024). *Implementasi Server Side Rendering Pada Sistem Travel Berbasis Website* (Vol. 4). https://santika.upnjatim.ac.id/submissions/index.php/santika/article/view/428

Putra, A. P., Sukadarmika, G., & Wiharta, D. M. (2024). Model Utilisasi Dan Visualisasi Resource Menggunakan Prometheus Dan Grafana Untuk Pengelolaan Server Di Universitas Udayana. *Majalah Ilmiah Teknologi Elektro*, *22*(2), 305. https://doi.org/10.24843/mite.2023.v22i02.p19

Rahman Hidayatullah, A., & Suartana, I. M. (2025). Penerapan Server Side Rendering untuk Meningkatkan Performa Website Kejaksaan Negeri X. *Journal of Informatics and Computer Science*, *07*. https://doi.org/10.26740/jinacs.v7n01.p12-20

Talluri, M. (2023). SEO Optimization for REST-Driven Angular Applications. In *Journal of Information Systems Engineering and Management* (Vol. 2023, Number 2). https://jisem-journal.com/

Vallamsetla, K. (2024). The Impact of Server-Side Rendering on UI Performance and SEO. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, *10*(5), 795–804. https://doi.org/10.32628/cseit241051067

Vepsäläinen, J. (2025). *Emergence of hybrid rendering models in web application development*. https://doi.org/10.13140/RG.2.2.36222.09282