Journal of Digital Innovation & Information Technology
Volume: 1, Nomor 1, Mei, 2024, *Hal*: 34-45

Analisis Performa Aplikasi Portal Portofolio Virtual Reality Berbasis Website Dengan Apache Benchmark Di PT Telekomunikasi Indonesia

Bayu Krisna[1], Ilham Saifudin[2], Lutfi Ali Muharom[3]

Program Studi Teknik Informatika, Universitas Muhammadiyah Jember
bayurollins.87@gmail.com, ilham.saifudin@unmuhjember.ac.id[2], lutfi.muharom@unmuhjember.ac.id[3]

Abstrak

Kinerja dan efisiensi aplikasi web sangat penting dalam memberikan pengalaman pengguna yang optimal. Penelitian ini berfokus pada analisis. Penelitian ini menggunakan Apache Benchmark untuk mengukur metrik performa seperti *Time per Request*, *Request per Second*, dan *Transfer Rate*. Pengujian dilakukan dengan membandingkan kinerja aplikasi saat menggunakan algoritma eager loading dan lazy loading dalam berbagai skenario beban kerja, termasuk jumlah *concurrent connection* yang berbeda. Hasil penelitian menunjukkan bahwa dalam kondisi *concurrent connection* rendah, algoritma *lazy loading* lebih unggul karena hanya memuat data yang diperlukan saat itu, sehingga lebih efisien dalam penggunaan sumber daya. Namun, saat *concurrent connection* meningkat, algoritma *eager loading* terbukti lebih efektif karena mampu menangani beban besar dengan lebih efisien melalui penggunaan *cache*. Pemilihan antara *eager loading* dan *lazy loading* harus mempertimbangkan karakteristik aplikasi dan beban kerja yang diharapkan. Dalam aplikasi portal portofolio *virtual reality* ini, algoritma *eager loading* dapat menjadi pilihan yang lebih baik untuk mengoptimalkan performa dan memberikan pengalaman pengguna yang lebih lancar.

Kata Kunci: *eager loading, lazy loading, request per second, transfer rate, time per request*

DOI: -
*Correspondensi: Bayu Krisna
Email: bayurollins.87@gmail.com

Received: 26 April 2024
Accepted: 21 Mei 2024
Published: 23 Mei 2024

Abstract

The performance and efficiency of web applications are crucial in providing an optimal user experience. This research focuses on analysis. This research uses Apache Benchmark to measure performance metrics such as Time per Request, Request per Second, and Transfer Rate. Testing is conducted by comparing the application's performance when using eager loading and lazy loading algorithms in various workload scenarios, including different numbers of concurrent connections. The research results indicate that under low concurrent connection conditions, the lazy loading algorithm excels as it only loads the necessary data at the time, making it more efficient in resource utilization. However, as concurrent connections increase, the eager loading algorithm proves to be more effective as it can handle large loads more efficiently through the use of cache. The choice between eager loading and lazy loading should consider the characteristics of the application and the expected workload. In this virtual reality portfolio portal application, the eager loading algorithm can be a better choice to optimize performance and provide a smoother user experience.

Keywords **:** *eager loading, lazy loading, request per second, transfer rate, time per request*

I. PENDAHULUAN

Dalam pengembangan aplikasi web seperti Aplikasi Portal Portofolio, kinerja dan efisiensi memiliki peran penting dalam memberikan pengalaman pengguna yang baik. Menurut penelitian "Pengujian Aplikasi Menggunakan Metode Load Testing dengan Apache Jmeter pada Sistem Informasi Pertanian" faktor - faktor dalam mengevaluasi kinerja *website* adalah *request per time, time per request, transfer rate*(Desy Intan Permatasari, 2020). Time per Request merupakan waktu yang diperlukan untuk menanggapi setiap permintaan yang diberikan oleh pengguna ke website. Semakin cepat website merespon permintaan, semakin baik juga pengalaman pengguna ketika mengakses website tersebut. Request per time merupakan jumlah permintaan yang dapat ditangani oleh website dalam satu waktu atau per detik.

Peneliti sebelumnya telah melakukan observasi di PT Telekomunikasi Indonesia dan telah mengembangkan Aplikasi Portofolio Virtual Reality Berbasis Website. Ketika aplikasi yang telah dikembangkan dilakukan pengujian menggunakan Apache Benchmark dengan endpoint yang diuji adalah all post mendapatkan hasil request per second 31.12 [#/sec], time per request 321.290 [ms] dan transfer rate 1940.86 [Kbytes/sec] dengan penentuan total request 100 dan concurrent connection 10. Dengan hasil pengujian yang dilakukan dengan Apache Benchmark tersebut, selanjutnya aplikasi yang telah dibuat akan mengimplementasikan algoritma agar mendapat hasil yang lebih baik dari sebelumnya .Terdapat dua algoritma yang akan digunakan untuk optimalisasi pemanggilan data di website tersebut yaitu eager loading dan lazy loading. Lazy loading hanya akan mengakses data yang diperlukan saat data tersebut dipanggil. Sementara eager loading mengakses seluruh data dalam database dengan satu perintah. Kedua metode tersebut merupakan komponen dalam ORM (Object Relational Mapping) pada framework Laravel(Ab - Apache HTTP Server Benchmarking Tool - Apache HTTP Server Version 2.4, n.d.; Otwel, n.d.). ORM merupakan Teknik Pemrograman yang menciptakan database berorientasi objek virtual yang dapat dimanipulasi dalam Bahasa pemrograman, sehingga setiap modifikasi, secara otomatis akan sinkron dengan database pada penyimpanan.

Penelitian Bire & Pawar(Bire & Pawar, 2021) dan Prasad(Prasad et al., 2021) menyoroti pentingnya optimasi kinerja aplikasi web. Bire & Pawar membandingkan aplikasi dengan dan tanpa lazy loading menggunakan Angular, menunjukkan bahwa lazy loading mempercepat pemrosesan awal (721 mikrodetik vs. 1,9 detik). Sementara itu, Prasad et al. fokus pada penerapan teknik eager loading pada basis data key-value, yang mengurangi jumlah query SQL, mempercepat waktu pemrosesan, dan meningkatkan responsivitas aplikasi. Kedua penelitian ini menekankan bahwa strategi optimasi yang berbeda dapat diterapkan untuk meningkatkan kinerja aplikasi web secara signifikan, baik melalui pemuatan konten yang lebih efisien maupun pengambilan data yang lebih cepat dari basis data.

Dengan studi kasus aplikasi portofolio berbasis website yang menampung data portofolio diharapkan dapat memberi gambaran terkait algoritma yang mampu menampilkan data dengan waktu dan proses lebih cepat menggunakan ORM pada framework Laravel. Penelitian tersebut diharapkan dapat menyelesaikan permasalahan terkait lamanya memuat halaman website yang memuat banyak data.

Website

Website adalah kumpulan halaman yang dirancang untuk menampilkan berbagai informasi, termasuk teks, gambar diam atau bergerak, animasi, dan elemen suara. Situs web ini dapat terdiri dari halaman-halaman yang bersifat statis atau dinamis, yang disusun secara terstruktur untuk memberikan pengalaman interaktif (Evans Fuad et al., 2021).

Laravel

Laravel adalah kerangka kerja (framework) aplikasi web berbasis PHP yang bersifat open-source. Framework ini menggunakan konsep Model-View-Controller (MVC) yang memudahkan pengembangan aplikasi web yang terstruktur dan mudah dikelola(Malang et al., 2019).

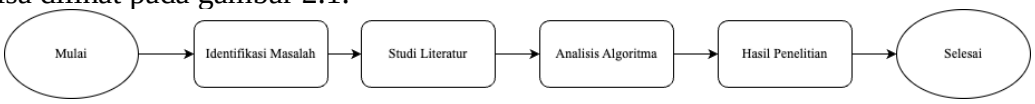
Apache Benchmark

Apache Benchmark adalah sebuah alat dari *Apache* yang dirancang untuk mengukur kinerja *website* atau server web. *Tool* tersebut menyediakan beberapa fitur *open source*, antarmuka baris perintah yang sederhana (*CLI*), serta kemampuan untuk melakukan pengujian beban dan kinerja(Apache, 2024).

II. METODE

2.1 Tahap Penelitian

Dalam proses penelitian ini ada beberapa tahapan yang akan dilakukan. *Flowchart* tahapan pengerjaan penelitian ini bisa dilihat pada gambar 2.1.



Gambar 2. 1 *Flowchar penelitian*

Dalam merancang Aplikasi Portofolio *Virtual Reality* Berbasis *Website* di PT Telkom Indonesia, penelitian ini memfokuskan pada pemilihan metode *loading* data, yaitu antara *eager loading* dan *lazy loading*. Identifikasi masalah yang terkait performa aplikasi *showcase virtual reality*. Dua algoritma *eager loading* dan *lazy loading*, menjadi hal utama dalam mengatasi masalah yang dihadapi. Selanjutnya adalah studi literatur. Studi literatur dapat dicari pada penelitian sebelumnya mengenai permasalahan ini lalu menentukan mana yang bisa dikembangkan dalam penelitian ini. Setelah dilakukannya studi literatur, maka dilakukan tahap analisis algoritma.

2.2 Identifikasi Masalah

Pada tahap identifikasi masalah, pada latar belakang telah dilakukan pengujian pada aplikasi portofolio tersebut. Dari hasil pengujian sebelumnya, bahwa pengambilan data masih bisa dioptimalkan lagi. Hasil dari identifikasi masalah ini menunjukan perlunya solusi dapat mengatasi masalah dalam menemukan algoritma yang sesuai untuk meningkatkan waktu memuat halaman *website*. Oleh karena itu, hal tersebut memotivasi pengembangan Aplikasi Portofolio *Virtual Reality* Berbasis Website dengan mengimplementasikan algoritma *eager loading* dan *lazy loading* pada *Object Relational Mapping* (ORM) Laravel. Dengan pemahaman yang lebih mendalam terkait identifikasi masalah tersebut, diharapkan penelitian ini dapat memberikan solusi yang efektif dan berkelanjutan, bertujuan untuk meningkatkan efisiensi *website* yang akan dibuat.

2.3 Studi Literatur

Metode studi literatur digunakan sebagai pendekatan untuk mengumpulkan data dan informasi yang mendukung penelitian. Penulis melakukan pembacaan pada sumber-sumber ilmiah yang terdapat di Google Scholar, seperti buku referensi, artikel, dan jurnal penelitian sebelumnya. Langkah tersebut bertujuan untuk memperoleh pemahaman mendalam tentang topik penelitian melalui literatur ilmiah yang dapat memperkaya landasan teoritis penelitian ini(Mualfah & Rajif, 2024).

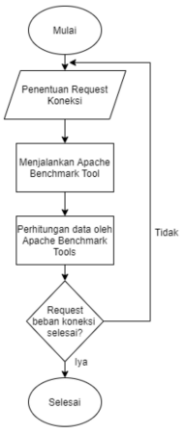
2.4 Analisis Algoritma

Sebelum ke tahap selanjutnya peneliti melakukan Analisa terhadapa algoritma *eager loading* dan *lazy loading. Eager Loading* adalah teknik yang digunakan dalam pemrograman berorientasi objek yang melibatkan inisialisasi objek dan struktur data sejak awal. Hal tersebut biasanya digunakan dalam konteks hubungan antara tabel dalam database(*Optimize Laravel Eloquent Queries with Eager Loading - Laravel News*, n.d.) *Lazy loading* adalah teknik pemrograman yang digunakan untuk menunda inisialisasi atau pengambilan objek, komputasi, atau operasi berat lainnya sampai saat dibutuhkan. Cara kerja algoritma ini adala dengan mengakses data relasi ketika dibutuhkan saja. *Lazy loading* adalah bagian dari teknik yang l disebut “pemuatan yang ditunda”, yang mencakup berbagai teknik untuk menunda operasi berat sampai mereka benar-benar diperlukan(*Lazy Loading - Web Performance | MDN*, 2023) cara kerja algoritma *eager loading* dan *lazy loading* dalam pemanggilan data di *website* bisa dilihat pada tabel 2.1

Tabel 2. 1 Cara kerja algoritma

Cara Kerja Algoritma	
Eager Loading	**Lazy Loading**
Algoritma akan mengidentifikasi relasi antara tabel yang ada dalam query. Hal tersebut bisa berupa one to many, many to one dari tabel yang ada dalam databse.	Ketika query dievaluasi. Hanya data utama yang dimuat dari database ke dalam memori aplikasi. Data relasi tidak dimuat pada itu juga.
Setelah relasi diidentifikasi, algoritma akan mengumpulkan data untuk setiap tabel yang berelasi dalam query. Hal tersebut dilakukan dalam satu pangilan ke database dan menambil data dari beberapa tabel sekaligus.	Saat aplikasi memnita akses ke data relasi yang belum dimuat, sistem akan mendeteksi permintaan tersebut.
Data yang terpanggil dimuat ke dalam memori aplikasi secara bersamaan. Sehingga seluruh data yang diperlukan dimuat kedalam memori aplikasi dalam satu operasi dari database.	Ketika dara relasi diminta, sistem secara dinamis memuat dari database ke dalam memori aplikasi. Ini dilakukan sekali ketika data tersebut diperlukan untuk menghindari pemboroasan memori dan kinerja yang tidak efisien.
Setelah data dimuat ke dalam memori, aplikasi dapat menggunakan data tersebut tanpa harus membuat panggilan database tambahan untuk tabel yang berelasi	

2.5 Studi Literatur



Gambar 2. 2 *Flowchart* penelitian

Pada gambar 2.2 merupakan alur pengujian sistem menggunakan Apache Benchmark Tools. Dalam tahap awal pengujian kinerja *website* proses diawali dengan penentuan *request* koneksi, proses pengujian *website* dilakukan dengan menentukan jumlah request (Acmad Valupi, 2021) yang ditunjukan pada tabel 2.2

Gambar 2. 3 Beban pengujian

Jumlah Koneksi	**Concurrent Connection**			**Kateori Pengujian**	
1000	10	50	100	Eager Loading	Lazy Loading
2000					
3000					

III. HASIL DAN PEMBAHASAN

3.1 Hasil

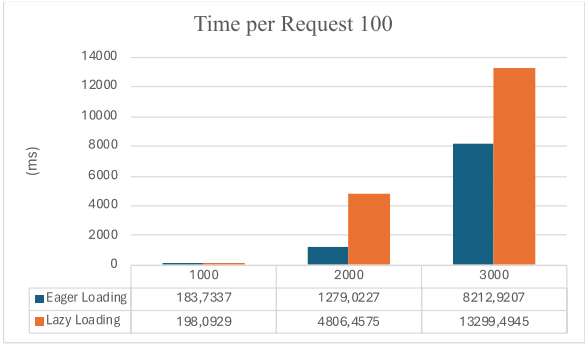
3.1.1 Time per Request

Gambar 3. 1 *Time per request* dengan 10 *concurrent connection*

Pada gambar 3.1 menunjukan hasil dari pengujian *time per request* untuk 10 *concurrent connection.* Pada 1000 beban *request*, nilai algoritma *eager loading* sebesar 161,3046 ms dan pada *lazy loading* sebesar 155,3939 ms. Dari kedua algoritma tersebut lazy memiliki waktu tercepat untuk menangani *request.* Pada 2000 beban request, nilai algoritma *eager loading* sebesar

Gambar 3. 2 *Time per request* dengan 50 *concurrent connection*

Pada gambar 3.2 menunjukan hasil dari pengujian *time per request* untuk 50 *concurrent connection.* Pada 1000 beban request, nilai algoritma *eager loading* sebesar 1194,4639 ms dan pada *lazy loading* sebesar 1013,6899 ms. Dari kedua algoritma tersebut lazy memiliki waktu tercepat untuk menangani *request.* Pada 2000 beban request, nilai algoritma *eager loading* sebesar 1017,2227 ms dan pada *lazy loading* sebesar 1097,1965 ms. Dari kedua algoritma tersebut *eager* memiliki waktu tercepat untuk menangani *request.* Pada 3000 beban request, nilai algoritma *eager loading* sebesar 548,36278 ms dan pada *lazy loading* sebesar 1279,0227 ms. Dari kedua algoritma tersebut *eager* memiliki waktu tercepat untuk menangani *request.*



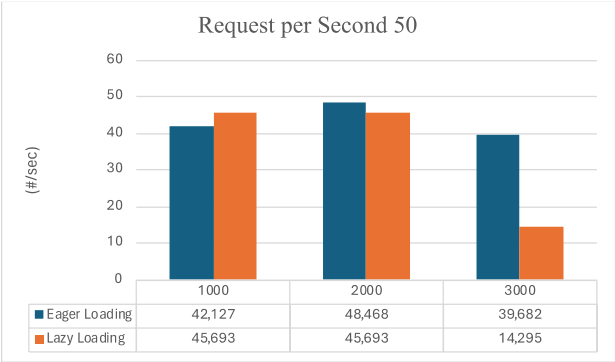
Gambar 3. 3 *Time per request* dengan 100 *concurrent connection*

Pada gambar 3.3 menunjukan hasil dari pengujian *time per request* untuk 100 *concurrent connection.* Pada 1000 beban request, nilai algoritma *eager loading* sebesar 183.7337 ms dan pada *lazy loading* sebesar 198,0929 ms. Dari kedua algoritma tersebut *eager* memiliki waktu tercepat untuk menangani *request.* Pada 2000 beban request, nilai algoritma *eager loading* sebesar 1279,0227 ms dan pada *lazy loading* sebesar 4806,4575 ms. Dari kedua algoritma tersebut *eager* memiliki waktu tercepat untuk menangani *request.* Pada 3000 beban request, nilai algoritma *eager loading* sebesar 8212,9207 ms dan pada *lazy loading* sebesar 13299,4945 ms. Dari kedua algoritma tersebut *eager* memiliki waktu tercepat untuk menangani *request.*

3.1.2 *Request per Second*

Gambar 3. 4 *Request per second* dengan 10 *concurrent connection*

Pada gambar 3.4 menunjukan hasil dari pengujian *request per second* untuk 10 *concurrent connection.* Pada 1000 beban request, nilai algoritma *eager loading* sebesar 62,031 / *second* dan pada *lazy loading* sebesar 64,691 / *second*. Dari kedua algoritma tersebut *lazy* menangani *request* terbanyak dari setiap detiknya. Pada 2000 beban request, nilai algoritma *eager loading* sebesar 58,662 / *second* dan pada *lazy loading* sebesar 64,691 / *second*. Dari kedua algoritma tersebut *lazy* menangani *request* terbanyak dari setiap detiknya. Pada 3000 beban request, nilai algoritma *eager loading* sebesar 50,606 / *second* dan pada *lazy loading* sebesar 54,65 / *second*. Dari kedua algoritma tersebut *lazy* menangani *request* terbanyak dari setiap detiknya.



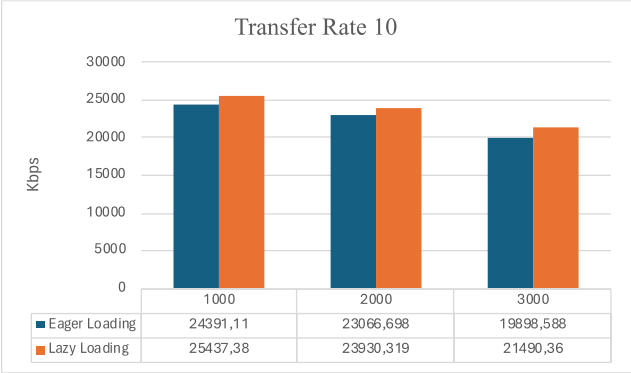
Gambar 3. 5 *Request per second* dengan 50 *concurrent connection*

Pada gambar 3.5 menunjukan hasil dari pengujian *request per second* untuk 50 *concurrent connection.* Pada 1000 beban request, nilai algoritma *eager loading* sebesar 42,127 / *second* dan pada *lazy loading* sebesar 45,693 / *second*. Dari kedua algoritma tersebut *lazy* menangani *request* terbanyak dari setiap detiknya. Pada 2000 beban request, nilai algoritma *eager loading* sebesar 48,468 / *second* dan pada *lazy loading* sebesar 45,693 / *second*. Dari kedua algoritma tersebut *eager* menangani *request* terbanyak dari setiap detiknya. Pada 3000 beban request, nilai algoritma *eager loading* sebesar 39,682 / *second* dan pada *lazy loading* sebesar 14,295 / *second*. Dari kedua algoritma tersebut *eager* menangani *request* terbanyak dari setiap detiknya.

Gambar 3. 6 *Request per second* dengan 100 *concurrent connection*

Pada gambar 3.6 menunjukan hasil dari pengujian *request per second* untuk 100 *concurrent connection.* Pada 1000 beban request, nilai algoritma *eager loading* sebesar 44,6 / *second* dan pada *lazy loading* sebesar 23,091 / *second*. Dari kedua algoritma tersebut *eager* menangani *request* terbanyak dari setiap detiknya. Pada 2000 beban request, nilai algoritma *eager loading* sebesar 36,381 / *second* dan pada *lazy loading* sebesar 13,144 / *second*. Dari kedua algoritma tersebut *eager* menangani *request* terbanyak dari setiap detiknya. Pada 3000 beban request, nilai algoritma *eager loading* sebesar 26,787 / *second* dan pada *lazy loading* sebesar 13,144 / *second*. Dari kedua algoritma tersebut *eager* menangani *request* terbanyak dari setiap detiknya.

3.1.3 *Transfer Rate*



Gambar 3. 7 *Transfer rate* dengan 10 *concurrent connection*

Pada gambar 3.7 menunjukan hasil dari pengujian *transfer rate* untuk 10 *concurrent connection.* Pada 1000 beban request, nilai algoritma *eager loading* sebesar 2439,11 Kbps dan pada *lazy loading* sebesar 25437,38 Kbps. Dari kedua algoritma tersebut *Lazy* paling cepat dalam pengiriman data. Pada 2000 beban request, nilai algoritma *eager loading* sebesar 23066,698 Kbps dan pada *lazy loading* sebesar 23930,319 Kbps. Dari kedua algoritma tersebut *Lazy* paling cepat dalam pengiriman data. Pada 3000 beban request, nilai algoritma *eager loading* sebesar 19898,588 Kbps dan pada *lazy loading* sebesar 21490,36 Kbps. Dari kedua algoritma tersebut *Lazy* paling cepat dalam pengiriman data.

Gambar 3. 8 ***Transfer rate*** **dengan 50 concurrent connection**

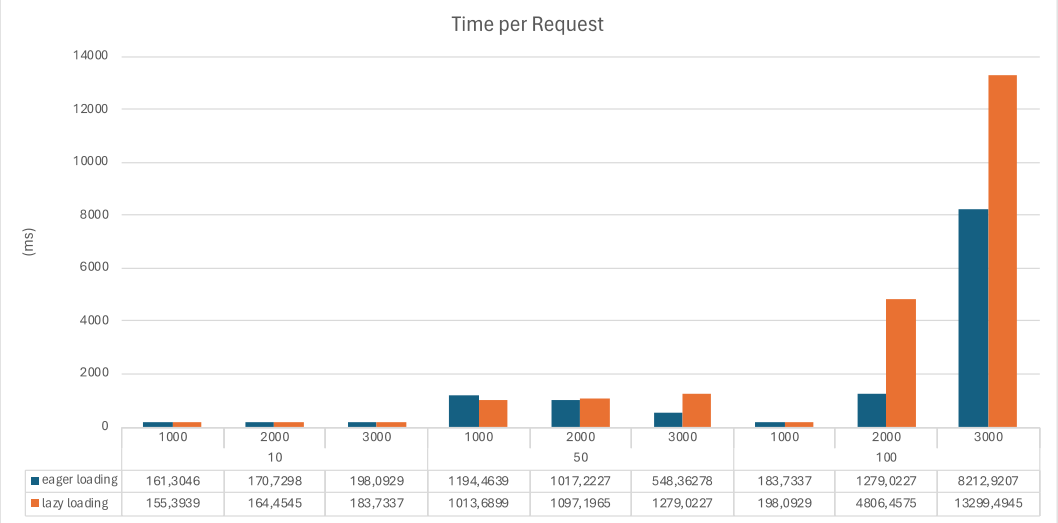
Pada gambar 3.8 menunjukan hasil dari pengujian *transfer rate* untuk 50 *concurrent connection.* Pada 1000 beban request, nilai algoritma *eager loading* sebesar 16564,784 Kbps dan pada *lazy loading* sebesar 23619,353 Kbps. Dari kedua algoritma tersebut *Lazy* paling cepat dalam pengiriman data. Pada 2000 beban request, nilai algoritma *eager loading* sebesar 23930,319 Kbps dan pada *lazy loading* sebesar 16441,367 Kbps. Dari kedua algoritma tersebut *Eager* paling cepat dalam pengiriman data. Pada 3000 beban request, nilai algoritma *eager loading* sebesar 15603,355 Kbps dan pada *lazy loading* sebesar 10620,636 Kbps. Dari kedua algoritma tersebut *eager* paling cepat dalam pengiriman data.

Gambar 3. 9 *Transfer rate* dengan 100 *concurrent connection*

Pada gambar 3.9 menunjukan hasil dari pengujian *transfer rate* untuk 100 *concurrent connection.* Pada 1000 beban request, nilai algoritma *eager loading* sebesar 17537,107 Kbps dan pada *lazy loading* sebesar 9079,702 Kbps. Dari kedua algoritma tersebut *eager* paling cepat dalam pengiriman data. Pada 2000 beban request, nilai algoritma *eager loading* sebesar 14343,997 Kbps dan pada *lazy loading* sebesar 5417,349 Kbps. Dari kedua algoritma tersebut *Eager* paling cepat dalam pengiriman data. Pada 3000 beban request, nilai algoritma *eager loading* sebesar 15603,355 Kbps dan pada *lazy loading* sebesar 5168,271 Kbps. Dari kedua algoritma tersebut *eager* paling cepat dalam pengiriman data.

3.2 Pembahasan

3.2.1 *Time per Request*

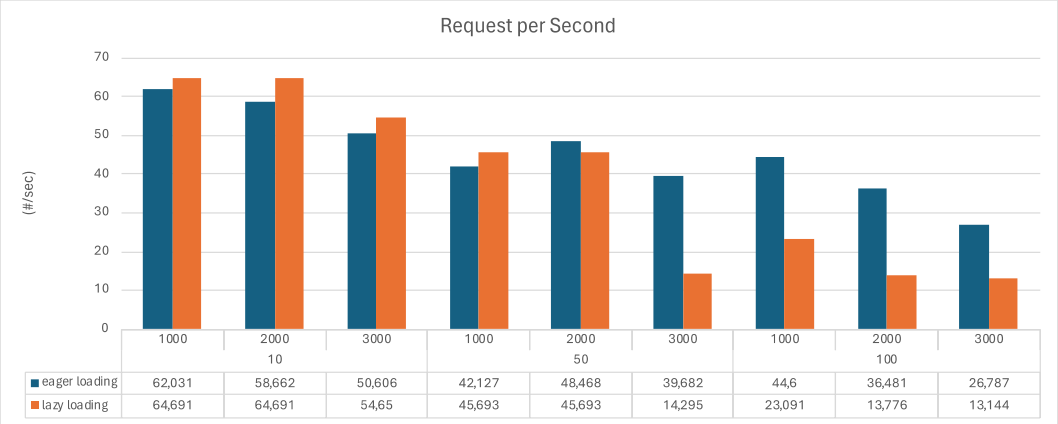


Gambar 3. 10 Grafik *time per request*

Pada gambar 3.10 menunjukan hasil dari pengujian *time per request* untuk 1000, 2000, 3000 total *request.* Pada kondisi 10 *concurrent connection* algoritma *lazy loading* lebih ungggul dari algoritma *eager loading.* Pada kondisi 50 *concurrent connection* mengalami sedikit kenaikan pada aloritma *eager loading* sehingga algoritma *eager loading* lebih unggul pada kondisi ini. Pada kondisi 100 *concurrent connection* kedua algoritma tersebut memiliki perbedaan yang signifikan dimana *eager loading* lebih cepat saat

menangani 1 buah *request* sedangkan *lazy loading* mengalami penurunan performa ketika menangani 1 buah *request*, sehingga algoritma *eager loading* lebih unggul pada kondisi ini

3.2.2 *Request per Second*

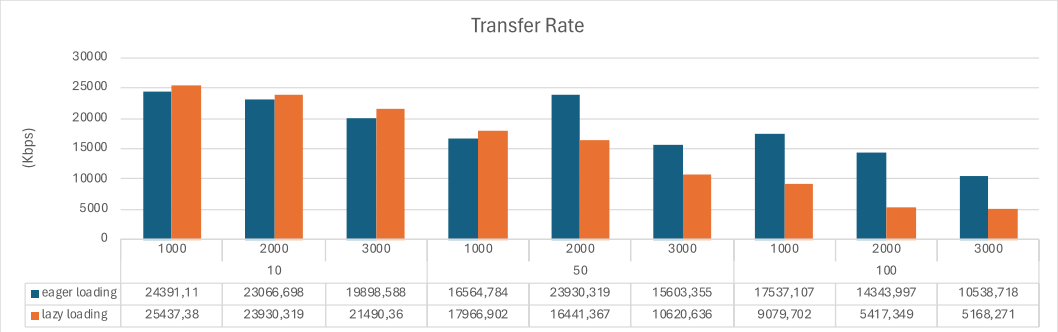


	1000	2000	3000	1000	2000	3000	1000	2000	3000
		10			50			100	
eager loading	62,031	58,662	50,606	42,127	48,468	39,682	44,6	36,481	26,787
lazy loading	64,691	64,691	54,65	45,693	45,693	14,295	23,091	13,776	13,144

Gambar 3. 11 Grafik *request per second*

Pada gambar 3.11 menunjukan hasil dari pengujian *request per second* untuk 1000, 2000, 3000 total *request.* Pada kondisi 10 *concurrent connection* algoritma *lazy loading* lebih ungggul dari algoritma *eager loading.* Pada kondisi 50 *concurrent connection* mengalami sedikit kenaikan pada aloritma *eager loading* sehingga algoritma *eager loading* lebih unggul pada kondisi ini. Pada kondisi 100 *concurrent connection* kedua algoritma tersebut memiliki perbedaan yang signifikan dimana *eager loading* memiliki performa lebih stabil ketika menangani banyaknya *request* dalam per detiknya, sedangkan *lazy loading* mengalami penurunan performa ketika menangani banyaknya *request dalam per detiknya* sehingga algoritma *eager loading* lebih unggul pada kondisi ini.

3.2.3 *Transfer Rate*



	1000	2000	3000	1000	2000	3000	1000	2000	3000
		10			50			100	
eager loading	24391,11	23066,698	19898,588	16564,784	23930,319	15603,355	17537,107	14343,997	10538,718
lazy loading	25437,38	23930,319	21490,36	17966,902	16441,367	10620,636	9079,702	5417,349	5168,271

Gambar 3. 12 Grafik *transfer rate*

Pada gambar 3.12 menunjukan hasil dari pengujian *transfer rate* untuk 1000, 2000, 3000 total *request.* Pada kondisi 10 *concurrent connection* algoritma *lazy loading* lebih ungggul dari algoritma *eager loading.* Pada kondisi 50 *concurrent connection* mengalami penurunan performa pada aloritma *lazy loading* sehingga algoritma *eager loading* lebih unggul pada kondisi ini. Pada kondisi 100 *concurrent connection* kedua algoritma tersebut memiliki perbedaan yang signifikan dimana *eager loading* memiliki kecepatan *transfer rate* lebih stabil sedangkan *lazy loading* mengalami penurunan kecepatan *transfer rate* pada *concurrent connection* yang lebih besar.

IV. KESIMPULAN

Kesimpulan yang didapat dari pengujian yang telah dilakukan adalah sebagai berikut:

1. Pada pengujian *Time per Request*, algoritma *lazy loading* memerlukan waktu lebih sedikit dalam menangani satu buah *request* pada *concurrent connection* rendah. Namun ketika menangani satu buah request pada *concurrent connection* lebih tinggi *lazy loading* memerlukan waktu lebih banyak dari *eager loading* dalam menangani satu buah *request.*
2. Pada pengujian *Request per Second*, nilai yang dihasilkan oleh algoritma *eager loading* relative lebih banyak dalam menangani *request* setiap detiknya. Hal tersebut membuktikan pemanfaatan *cache* dari algoritma *eager loding*
3. Pada pengujian *Transfer Rate*, algoritma *eager loading* memiliki kecepatan relative tinggi dari *lazy loading*. *Eager loading* mampu mempertahankan performanya ketika menangani *concurrent connection* yang tinggi dari pada *lazy loading*

Dari kedua algoritma tersebut, algoritma *eager loading* lebih optimal digunakan pada *website* yang memiliki trafik lebih tinggi dengan *concurrent connection* yang tinggi. Sedangkan algoritma *lazy loading* lebih optimal digunakan pada *website* yang memiliki trafik lebih rendah dengan *concurrent connection* yang rendah

DAFTAR PUSTAKA

ab - Apache HTTP server benchmarking tool - Apache HTTP Server Version 2.4. (n.d.). Retrieved May 6, 2024, from https://httpd.apache.org/docs/2.4/programs/ab.html

Acmad Valupi. (2021). *ANALISIS PERFORMANSI WEB SERVER APACHE, NGINX DAN OPENLITESPEED BERBASIS CONTAINER PADA VPS PERFORMANCE ANALYSIS APACHE, NGINX AND OPENLITESPEED WEB SERVER BASED ON CONTAINER ON VPS*.

Bire, S., & Pawar, V. (2021). Lazy Loading Based with Load On Demand and Currency Support in Web Browser. *International Journal of Scientific Research in Science, Engineering and Technology*, 473–478. https://doi.org/10.32628/ijsrset2183183

Desy Intan Permatasari, M. A. , A. Y. M. N. I. S. G. P. S. R. D. A. N. W. N. (2020). *Pengujian Aplikasi Menggunakan Metode Load Testing dengan Apache Jmeter pada Sistem Informasi Pertanian*.

Evans Fuad, Regiolina Hayami, & Kharisma, A. (2021). Evaluasi Usabilitas Website E-Learning Umri Terhadap Mahasiswa Umri Menggunakan Metode Usability Testing. *Jurnal CoSciTech (Computer Science and Information Technology)*, *2*(2), 74–82. https://doi.org/10.37859/coscitech.v2i2.3029

Lazy loading - Web performance | MDN. (2023, December 20). https://developer.mozilla.org/en-US/docs/Web/Performance/Lazy_loading

Malang, P. K., Informatika, T., & Malang, P. K. (2019). *IMPLEMENTASI FRAMEWORK LARAVEL DALAM SISTEM*. *2*(2), 35–42.

Mualfah, D., & Rajif, I. (2024). Analisis dan perancangan user interface dan user experience untuk sistem asuransi karyawan berbasis web menggunakan metode design thinking. *Jurnal CoSciTech (Computer Science and Information Technology)*, *4*(3), 686–695. https://doi.org/10.37859/coscitech.v4i3.6233

Optimize Laravel Eloquent Queries with Eager Loading - Laravel News. (n.d.). Retrieved February 27, 2024, from https://laravel-news.com/eloquent-eager-loading

Otwel, T. (n.d.). *Eloquent: Getting Started - Laravel 9.x - The PHP Framework For Web Artisans*. Retrieved May 11, 2024, from https://laravel.com/docs/9.x/eloquent

Prasad, R., Panigrahi, M. B. K., Kaushik, B. K., & Roy, S. (2021). *Lecture Notes in Networks and Systems 177 Proceedings of 6th International Conference on Recent Trends in Computing*. http://www.springer.com/series/15179