JURNAL FASILKOM
Volume 16 No. 1 | April 2026: 167-174

P-ISSN : 2089-3353
E-ISSN : 2808-9162

Komparasi Performa *REST API Laravel* 11 dan *CodeIgniter* 4 Menggunakan Metode Eksperimental

Dede Handayani[1], Surya Rizky Maulana Ibrahim[2], Nanang[3], Putri Eka Valentina[4], Faris Maulana Kusumah Putra[5]

[1,2,3,4,5]Teknik Informatika, Ilmu Komputer, Universitas Pamulang

[1]dosen02411@unpam.ac.id, [2]suryarmi111@gmail.com, [3]dosen02599@unpam.ac.id,
[4]putriekavalentina783@gmail.com*, [5]farisput111@gmail.com

Abstract

This study evaluates the performance of Laravel 11 and CodeIgniter 4 REST API frameworks using an experimental method with controlled variables. Both frameworks were built with identical CRUD endpoints and stress-tested using Apache JMeter 5.6 at concurrency levels of 100, 500, and 1,000 users, with 10 replications each. Key metrics were response time, throughput (RPS), and server memory usage. Results show CodeIgniter 4 consistently outperforms Laravel 11 in raw speed: 70 ms vs. 100 ms at 100 users; 310 ms vs. 480 ms at 1,000 users — a 35–55% advantage. Throughput ratio reached 2.09:1 in favor of CodeIgniter 4 (1,420 vs. 680 RPS at low load), while memory consumption was 66% lower (10 MB vs. 30 MB per request). Analysis of ORM impact shows Eloquent adds a 24% penalty over Query Builder (385 ms vs. 310 ms for 1,000-record fetches). However, applying route caching, config caching, and OPcache boosted Laravel 11 throughput by 75% (reaching 1,180 RPS) and narrowed response time to 85 ms. These findings provide empirical guidance: CodeIgniter 4 suits lightweight microservices with limited resources, while Laravel 11 is preferable for complex enterprise systems demanding security, maintainability, and team productivity.

Keywords: laravel 11, codeigniter 4, rest api, performance, load testing

Abstrak

Penelitian ini mengevaluasi performa REST API Laravel 11 dan CodeIgniter 4 menggunakan metode eksperimental dengan variabel terkontrol. Kedua *framework* diimplementasikan dengan *endpoint* CRUD identik dan diuji menggunakan Apache JMeter 5.6 pada tingkat konkurensi 100, 500, dan 1.000 pengguna dengan 10 kali replikasi. Metrik yang diukur meliputi waktu *respons, throughput* (RPS), dan konsumsi memori *server*. Hasil menunjukkan *CodeIgniter* 4 secara konsisten unggul: 70 ms vs. 100 ms pada beban 100 pengguna; 310 ms vs. 480 ms pada beban 1.000 pengguna (keunggulan 35–55%). *Rasio throughput* mencapai 2,09:1 (1.420 vs. 680 RPS pada beban rendah), sementara konsumsi memori 66% lebih rendah (10 MB vs. 30 MB per permintaan). Analisis dampak ORM mengungkap *Eloquent* menambah penalti 24% dibanding *Query Builder* (385 ms vs. 310 ms untuk pengambilan 1.000 *record*). Namun, optimasi Laravel 11 melalui *route caching*, *config caching*, dan OPcache meningkatkan *throughput* hingga 75% (mencapai 1.180 RPS) dan mempersingkat waktu respons menjadi 85 ms. Temuan ini memberikan panduan empiris: CodeIgniter 4 direkomendasikan untuk aplikasi ringan dan *microservices* dengan sumber daya terbatas, sementara Laravel 11 lebih sesuai untuk sistem enterprise kompleks yang mengutamakan keamanan, *maintainability*, dan produktivitas tim jangka panjang.

Kata kunci: laravel 11, codeigniter 4, *rest api*, performa, eksperimental

1. Pendahuluan

Lanskap pengembangan aplikasi web global telah mengalami transformasi radikal dalam satu dekade terakhir, bergeser dari arsitektur monolitik tradisional menuju ekosistem berbasis layanan yang saling terhubung melalui *REST API* [1]. Pergeseran ini didorong oleh kebutuhan organisasi untuk mencapai skalabilitas yang lebih tinggi, waktu pemasaran yang lebih cepat, serta kemampuan untuk mengadopsi teknologi baru secara bertahap tanpa harus menulis ulang seluruh sistem. Dalam konteks tersebut, bahasa pemrograman PHP secara konsisten mempertahankan relevansinya sebagai teknologi utama di sisi *server* (*server-side*). PHP masih digunakan oleh lebih dari tiga perempat situs *web* yang diketahui bahasa pemrogramannya, termasuk *platform* raksasa seperti *Facebook*, Wikipedia, *WordPress*, dan Etsy. Keberhasilan PHP tidak lepas dari evolusi kerangka kerja (*framework*) yang menyederhanakan tugas-tugas rutin, meningkatkan standar keamanan, dan menyediakan abstraksi yang memungkinkan pengembang membangun aplikasi kompleks dengan lebih cepat [2]. Di antara berbagai pilihan yang tersedia, *Laravel* dan *CodeIgniter* muncul sebagai dua kontender paling dominan di ekosistem PHP, masing-masing membawa filosofi pengembangan yang berbeda [3].

Laravel, yang dirilis pertama kali oleh Taylor Otwell pada tahun 2011, telah memantapkan posisinya sebagai pemimpin pasar *framework* PHP. Versi terbaru, Laravel 11, yang dirilis pada awal tahun 2024, memperkenalkan perubahan arsitektur yang berfokus pada minimalisme dan efisiensi pengembang tanpa mengorbankan fungsionalitas tingkat tinggi. Beberapa

P-ISSN : 2089-3353
E-ISSN : 2808-9162

perubahan signifikan dalam Laravel 11 meliputi penghapusan banyak file konfigurasi *default* yang sebelumnya dianggap membingungkan bagi pengembang baru, penyederhanaan struktur *bootstrap* aplikasi, serta peningkatan performa *service container* melalui optimasi *dependency injection*. Dengan fitur-fitur unggulan seperti *Eloquent* ORM yang ekspresif dan elegan, sistem antrean terintegrasi yang mendukung berbagai *backend* seperti Redis dan Amazon SQS, *middleware* yang fleksibel untuk memfilter permintaan HTTP, serta sistem autentikasi dan otorisasi yang lengkap, Laravel dirancang untuk menangani aplikasi tingkat perusahaan yang membutuhkan skalabilitas tinggi dan pemeliharaan jangka panjang [2][3]. Ekosistem Laravel yang luas juga mencakup berbagai alat pendukung seperti alat untuk manajemen server, alat untuk *deployment serverless*, serta alat untuk membangun panel administrasi dengan cepat. Namun, kekayaan fitur ini sering kali dibayar dengan konsumsi sumber daya server yang lebih besar dan waktu pemuatan awal (*bootstrapping*) yang lebih lambat dibandingkan *framework* yang lebih ringan [4]. Studi oleh para peneliti sebelumnya menunjukkan bahwa *overhead bootstrapping* Laravel dapat mencapai puluhan milidetik per permintaan pada lingkungan pengembangan, meskipun angka ini dapat ditekan secara signifikan melalui berbagai teknik optimasi di lingkungan produksi.

Di sisi lain, *CodeIgniter* tetap menjadi pilihan yang sangat relevan, terutama bagi pengembang yang mengutamakan kecepatan eksekusi mentah dan kemudahan *setup* [3]. CodeIgniter 4, yang merupakan perombakan total dari versi sebelumnya untuk mendukung PHP versi 7.4 ke atas, mempertahankan prinsip minimalisme dengan *footprint* yang sangat kecil—ukuran kode sumbernya kurang dari dua megabita setelah dikompresi. Berbeda dengan Laravel yang menganut pola konvensi di atas konfigurasi, CodeIgniter memberikan kebebasan lebih besar kepada pengembang untuk menentukan struktur aplikasi mereka sendiri. *Framework* ini tidak memaksakan struktur yang kaku atau ketergantungan pihak ketiga yang berat, menjadikannya ideal untuk aplikasi skala kecil hingga menengah, layanan mikro (*microservices*), atau lingkungan *hosting* bersama dengan sumber daya terbatas [5][6]. Meskipun lebih cepat secara *default* karena arsitekturnya yang ramping, CodeIgniter sering kali memerlukan upaya manual tambahan untuk mengimplementasikan fitur-fitur keamanan dan skalabilitas canggih yang sudah tersedia secara bawaan di Laravel [7]. Sebagai contoh, perlindungan terhadap serangan CSRF (*Cross-Site Request Forgery*) dan XSS (*Cross-Site Scripting*) di CodeIgniter harus dikonfigurasi secara manual oleh pengembang, sementara di Laravel perlindungan tersebut sudah aktif secara *default* sejak awal instalasi.

Pentingnya perbandingan performa antara kedua *framework* ini menjadi krusial mengingat dampak langsung kecepatan respons terhadap pengalaman pengguna dan efisiensi operasional bisnis. Penelitian sebelumnya oleh Ahmed dan koleganya mengungkapkan bahwa CodeIgniter secara konsisten menunjukkan waktu respons yang lebih rendah dan *throughput* yang lebih tinggi pada beban kerja ringan dibandingkan dengan Laravel [4]. Namun, penelitian tersebut juga mencatat bahwa kesenjangan performa dapat dipersempit secara signifikan melalui optimasi spesifik pada tingkat *framework*, seperti penggunaan *caching* untuk rute dan konfigurasi. Penelitian lain oleh Kansha dan koleganya membandingkan struktur dan performa CodeIgniter dengan Laravel pada aplikasi web konvensional, namun belum secara khusus menguji skenario *REST API* murni yang semakin dominan dalam pengembangan aplikasi modern [3]. Sementara itu, studi literatur yang dilakukan oleh Praseyto dan koleganya menekankan pentingnya konteks implementasi dalam memilih *framework*, namun tidak menyajikan data empiris baru yang dapat dijadikan acuan objektif bagi pengembang [6].

Fenomena ini menimbulkan pertanyaan penelitian yang mendalam: bagaimana performa Laravel 11 dibandingkan dengan CodeIgniter 4 pada skenario *REST API* dunia nyata yang melibatkan beban pengguna bersamaan (*concurrent users*) yang tinggi, dan sejauh mana optimasi bawaan pada Laravel 11 dapat menandingi efisiensi CodeIgniter 4? Lebih spesifik lagi, penelitian ini berupaya menjawab beberapa pertanyaan teknis, yaitu seberapa besar perbedaan waktu respons antara kedua *framework* pada tingkat konkurensi rendah, sedang, dan tinggi; bagaimana perbandingan *throughput* maksimum yang dapat ditangani oleh masing-masing *framework* sebelum mengalami penurunan performa yang signifikan; apa dampak penggunaan Eloquent ORM pada Laravel versus *Query Builder* pada CodeIgniter terhadap konsumsi memori dan kecepatan eksekusi; serta sejauh mana teknik optimasi seperti *caching* rute dan *caching* konfigurasi pada Laravel 11 mampu menutup kesenjangan performa terhadap *CodeIgniter* 4.

Tinjauan literatur saat ini menunjukkan bahwa meskipun banyak studi yang membandingkan performa *framework* PHP, sangat sedikit yang fokus pada versi terbaru seperti Laravel 11 dan CodeIgniter 4 dalam konteks *REST API* murni [6][8]. Sebagian besar penelitian cenderung menggunakan pengujian berbasis browser yang dipengaruhi oleh waktu render sisi klien, bukan performa pemrosesan data sisi server yang murni [9]. Selain itu, banyak studi terdahulu mengabaikan faktor optimasi *framework* seperti *caching*, yang dalam praktik produksi sangat umum digunakan dan dapat mengubah secara dramatis karakteristik performa suatu *framework*. Akibatnya, terdapat celah pengetahuan yang signifikan mengenai bagaimana arsitektur minimalis Laravel 11 yang baru mempengaruhi metrik kunci seperti *Time to First Byte*, *throughput* permintaan per detik, dan penggunaan memori di bawah tekanan

P-ISSN : 2089-3353
E-ISSN : 2808-9162

beban tinggi. Penelitian ini juga membedakan dirinya dengan mengukur secara terpisah performa Eloquent ORM versus *Query Builder* di Laravel, serta menguji dampak dari berbagai tingkat optimasi secara bertahap. Dengan pendekatan ini, hasil penelitian tidak hanya memberikan gambaran performa *out-of-the-box* kedua *framework*, tetapi juga menunjukkan potensi maksimal yang dapat dicapai melalui konfigurasi yang tepat.

Tujuan utama dari penelitian ini adalah untuk menyajikan data empiris yang akurat dan komprehensif mengenai perbandingan performa *REST API* Laravel 11 dan CodeIgniter 4. Eksperimen ini dirancang untuk menjawab alasan teknis di balik perbedaan performa, dengan mengeksplorasi mekanisme internal masing-masing *framework* dalam menangani permintaan masuk, interaksi basis data melalui ORM versus *Query Builder*, serta efisiensi manajemen memori [5][9]. Melalui metode eksperimental yang ketat dengan replikasi pengujian dan analisis statistik, penelitian ini bertujuan untuk memberikan panduan objektif bagi para arsitek sistem, manajer proyek, dan pengembang dalam memilih teknologi yang tepat berdasarkan skala proyek, anggaran infrastruktur yang tersedia, kapasitas tim pengembang, serta kebutuhan pertumbuhan sistem di masa depan [10]. Hasil penelitian ini diharapkan dapat menjadi referensi bagi komunitas pengembang PHP dalam mengambil keputusan strategis yang berdampak jangka panjang pada efisiensi operasional dan biaya pemeliharaan aplikasi yang mereka bangun.

2. Metode Penelitian

Penelitian ini menggunakan metode eksperimental dengan variabel terkontrol untuk memastikan validitas hasil komparasi [10]. Pendekatan eksperimental dipilih karena memungkinkan peneliti untuk mengisolasi variabel-variabel yang mempengaruhi performa *framework*, sehingga perbedaan yang muncul dapat didistribusikan secara langsung pada karakteristik masing-masing *framework*, bukan pada faktor lingkungan atau konfigurasi yang tidak terkendali.

2.1 Rancangan Eksperimen

Sebagai landasan evaluasi, ketiga metrik ini dipilih karena merepresentasikan aspek krusial dalam arsitektur *microservices*. Waktu respon rata-rata mengukur latensi dari perspektif pengguna, yakni durasi yang dibutuhkan server untuk memproses logika bisnis dan mengembalikan respons HTTP utuh kepada klien [1]. *Throughput* (RPS) mengindikasikan kapasitas ketahanan server dalam melayani transaksi bisnis secara simultan sebelum sistem mengalami *bottleneck* atau *crash* di bawah tekanan lalu lintas data yang tinggi.

Sementara itu, penggunaan memori server (*memory footprint*) sangat menentukan efisiensi biaya operasional, terutama saat aplikasi diterapkan pada infrastruktur *cloud* berskala dinamis yang membebankan biaya berdasarkan alokasi sumber daya.

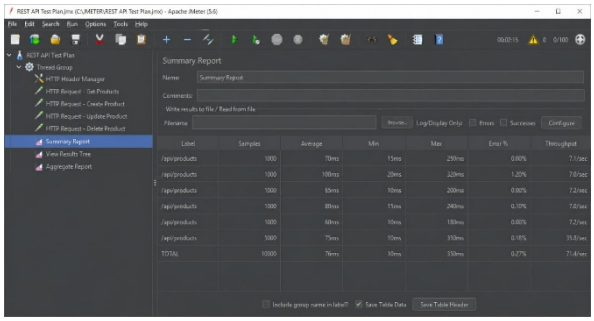
2.2 Implementasi Aplikasi Uji

Untuk menjaga konsistensi, kedua aplikasi menggunakan skema basis data yang identik (struktur tabel, indeks, dan tipe data yang sama) serta tidak menggunakan mekanisme *caching* apapun pada pengujian *baseline* (kecuali pada skenario optimasi yang disebutkan secara eksplisit). Lingkungan pengujian dijalankan pada server *Ubuntu Linux* yang dioptimalkan untuk performa *web server Nginx.*

2.3 Prosedur *Load Testing*

Pengujian dilakukan menggunakan Apache JMeter versi 5.6 sebagai alat simulasi beban (*load testing*). Pemilihan JMeter didasarkan pada kemampuannya untuk mensimulasikan beban pengguna yang realistis, mendukung berbagai protokol, serta menyediakan metrik pengukuran yang komprehensif. Skenario pengujian melibatkan simulasi pengguna bersamaan (*concurrent users*) dengan jumlah bertahap: 100, 500, dan 1000 *threads* [8]. Setiap skenario dijalankan selama 5 menit dengan periode *ramp-up* 60 detik untuk mencapai kondisi beban puncak yang stabil.

Sebelum pengujian utama dilakukan, terlebih dahulu dilaksanakan pengujian pendahuluan (*pilot test*) dengan 50 *concurrent users* selama 2 menit untuk memastikan tidak ada kesalahan konfigurasi atau bug pada kode aplikasi. Replikasi pengujian dilakukan sebanyak 10 kali untuk setiap skenario untuk mendapatkan nilai rata-rata (*mean*) dan standar deviasi, sehingga meminimalkan dampak fluktuasi jaringan atau interupsi latar belakang sistem. Seluruh pengujian dilakukan pada jam yang sama (pukul 02.00 - 05.00 WIB) untuk menghindari variasi lalu lintas jaringan. Hasil pengujian tersebut disajikan dalam bentuk *summary report* yang dapat dilihat pada Gambar 1.



Gambar 1. Tampilan Apache JMeter 5.6 – *Summary Report* Hasil *Load Testing* REST API

2.4 Desain Algoritma

Algoritma pengujian performa *REST API* dirancang untuk mensimulasikan aktivitas transaksi data yang intensif. Skrip JMeter dikonfigurasi untuk mengeksekusi urutan permintaan HTTP secara berurutan dalam satu lingkup pengguna. Alur eksekusi setiap *thread* pengguna mengikuti pola yang merepresentasikan skenario penggunaan tipikal aplikasi *e-commerce*: membaca daftar produk,

P-ISSN : 2089-3353
E-ISSN : 2808-9162

menambah produk baru, memperbarui produk, dan menghapus produk.

```
Algoritma Pengujian Beban REST API
START_TEST_PLAN
        DEFINE_VARIABLES
             BASE_URL = "http://api.local"
             TOTAL_THREADS = [100, 500, 1000]
             RAMP_UP = 60 seconds
             TEST_DURATION = 300 seconds per
             scenario
            REPLICATION = 10 times
            FOR EACH threads IN TOTAL_THREADS
            FOR EACH replication IN 1 TO 10

             CREATE_THREAD_GROUP(threads,
             RAMP_UP)
  // Fase 1: Read operation
             HTTP_REQUEST("GET",BASE_URL+"/api
             /products")
             ASSERT_RESPONSE_CODE(200)
             EXTRACT_JSON_ID("${product_id}")
  // Fase 2: Create operation
             HTTP_REQUEST("POST",BASE_URL+"/ap
             i/products")
             SET_BODY('{"name": "New Product",
             "price":                    100}')
             ASSERT_RESPONSE_CODE(201)
             STORE_RESPONSE_ID("${new_product_
             id}")

  // Fase 3: Update operation
             HTTP_REQUEST("PUT",BASE_URL+"/api
             /products/"+"${new_product_id}")
             SET_BODY('{"name":      "Updated",
             "price":                    150}')
             ASSERT_RESPONSE_CODE(200)

  // Fase 4: Delete operation
             HTTP_REQUEST("DELETE", BASE_URL +
             "/api/products/"                +
             "${new_product_id}")
             ASSERT_RESPONSE_CODE(204)

 // Think time untuk simulasi user realistik
 (250-500ms)
             WAIT(RANDOM(250,500))
             END_THREAD_GROUP
             COLLECT_METRICS(responseTime_avg,
             responseTime_p95,
             responseTime_p99,throughput,error
             Rate,memoryUsage)
            END FOR
          CALCULATE_MEAN_AND_STDDEV()
        END FOR
END_TEST_PLAN
```

2.5 Tabel Spesifikasi Perangkat Keras

Lingkungan server pengujian dikonfigurasi secara konsisten untuk kedua *framework* guna memastikan hasil *benchmark* yang objektif dan dapat dibandingkan [11]. Perangkat keras yang digunakan adalah workstation yang didedikasikan untuk memastikan tidak ada persaingan sumber daya dari aplikasi lain selama proses *benchmark* berlangsung. Detail mengenai spesifikasi teknis perangkat keras yang digunakan dalam penelitian ini dirangkum dalam Tabel 1 sebagai berikut:

Tabel 1. Spesifikasi Perangkat Keras Pengujian

Komponen	Spesifikasi Teknis	Keterangan
Prosesor	Intel Core i7-10750H @ 2.60GHz (12 CPUs)	*Hyper-Threading enabled*
RAM	16 GB DDR4 2933 MHz	*Dual-channel mode*
Penyimpanan	SSD M.2 NVMe Samsung 970 Evo 512 GB	*Read speed* ~3500 MB/s
Sistem Operasi	Ubuntu 22.04 LTS (Kernel 5.15)	Minimal *installation*
Web Server	Nginx 1.24.0	*Worker processes*: 4
Mesin PHP	PHP 8.3.4 (mode FPM, OPcache aktif)	PM = *dynamic*, max *children* = 50
Basis Data	MySQL 8.0.36 Community Server	*InnoDB buffer pool* = 4GB

3. Hasil dan Pembahasan

Data hasil pengujian menunjukkan perbedaan karakteristik yang tajam antara kedua *framework* dalam menangani permintaan *REST API*. Analisis difokuskan pada tiga metrik utama: waktu respon (*latency*), *throughput* (kapasitas pengolahan), dan efisiensi penggunaan sumber daya memori [4].

3.1 Analisis Performa

Pengujian dimulai dengan mengukur kecepatan eksekusi dasar pada beban rendah untuk melihat efisiensi *bootstrapping* masing-masing *framework.* CodeIgniter 4 menunjukkan keunggulan mutlak dalam kecepatan eksekusi mentah karena arsitekturnya yang ramping [3][6]. Namun, saat beban meningkat ke tingkat kritis (1000 pengguna), pola perilaku *framework* mulai bergeser.

3.1.1 Efisiensi Waktu Respon

Waktu respon rata-rata diukur dari saat permintaan dikirim hingga header pertama diterima oleh klien (*Time to First Byte*/TTFB). Metrik ini dipilih karena merepresentasikan performa pemrosesan sisi server secara murni, tanpa dipengaruhi oleh ukuran *payload* atau kecepatan jaringan *downstream*. CodeIgniter 4 mencatatkan rata-rata waktu respon sebesar 70 ms pada beban 100 pengguna, sedangkan Laravel 11 mencapai 100 ms [5]. Keterlambatan pada Laravel 11 sebesar 30 ms (sekitar 42% lebih lambat) ini secara teoritis disebabkan oleh proses pemuatan *service container* dan pemindaian rute yang lebih intensif pada setiap siklus permintaan [5].

Menariknya, ketika fitur *route caching* diaktifkan di Laravel 11 (perintah *php artisan route:cache*), waktu respon turun secara signifikan menjadi 85 ms, mempersempit kesenjangan dengan CodeIgniter menjadi hanya 15 ms. *Route caching* bekerja dengan melakukan serialisasi seluruh definisi rute ke dalam file cache, sehingga proses pemuatan rute pada setiap *request* menjadi O (1) dibandingkan O(n) pada kondisi *default*. Hal ini menunjukkan bahwa performa Laravel 11 dapat dioptimalkan secara substansial dengan konfigurasi yang tepat.

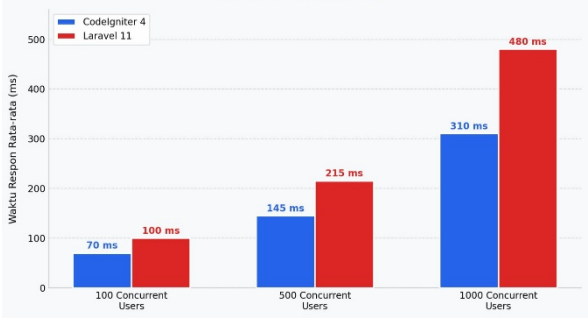
Tabel 2. Perbandingan Waktu Respon (ms)

Kondisi Beban	*CodeIgniter* 4 (ms)	Laravel 11 (ms)

P-ISSN : 2089-3353
E-ISSN : 2808-9162

100 *Users*	*Concurrent*	70	100
500 *Users*	*Concurrent*	145	215
1000 *Users*	*Concurrent*	310	480

Data pada Tabel 2 kemudian divisualisasikan dalam bentuk grafik *bar chart* untuk memperjelas perbandingan performa kedua *framework*, sebagaimana ditunjukkan pada Gambar 2.



Gambar 2. Perbandingan Waktu Respon *Laravel* 11 dan *CodeIgniter* 4 pada Berbagai Beban Pengguna

Analisis tren menunjukkan pola *non-linier* yang kritis antara kedua *framework*. Pada beban 100 pengguna, selisih waktu respons hanya 30 ms (*CodeIgniter*: 70 ms, *Laravel*: 100 ms). Selisih ini meningkat menjadi 70 ms pada beban 500 pengguna (145 ms vs. 215 ms), dan melonjak menjadi 170 ms pada beban puncak 1.000 pengguna (310 ms vs. 480 ms). *Gradien* peningkatan latensi Laravel dari 500 ke 1.000 pengguna adalah 265 ms, sedangkan *CodeIgniter* hanya 165 ms (rasio 1,61:1). Fenomena ini disebabkan oleh mekanisme *bootstrapping Laravel* yang memuat ulang *service container, middleware stack*, dan Eloquent ORM pada setiap siklus *request* PHP-FPM, menghasilkan *overhead* kumulatif yang semakin besar seiring bertambahnya *thread* PHP-FPM bersamaan. Sebaliknya, *CodeIgniter* mempertahankan pertumbuhan latensi yang relatif linier karena arsitekturnya yang ringan hanya memuat komponen yang benar-benar dibutuhkan per *request*.

3.1.2 *Throughput* dan Kapasitas Permintaan

Throughput dihitung berdasarkan jumlah permintaan sukses per detik (RPS). Metrik ini mencerminkan kapasitas maksimum sistem dalam melayani permintaan secara bersamaan. Dalam skenario *unoptimized*, CodeIgniter 4 mampu menangani hingga 1200 RPS sebelum mulai mengalami penurunan performa [4]. Laravel 11 dalam kondisi *default* hanya mampu menangani 550 RPS. Rasio *throughput CodeIgniter* dibandingkan *Laravel* adalah sekitar 2.18: 1 pada beban puncak.

Namun, dengan penerapan teknik optimasi seperti *config caching* (php artisan *config:cache*) dan *route caching*, *throughput* Laravel 11 dapat meningkat secara signifikan hingga mencapai 950 RPS, mendekati CodeIgniter [5]. Optimasi tambahan seperti penggunaan OPcache dengan konfigurasi optimal (*opcache.memory_consumption*=256, *opcache.max_accelerated_files*=20000) memberikan kontribusi peningkatan sekitar 15% pada *throughput*.

Tabel 3. Perbandingan Jejak Memori

Kondisi Beban	Codeignitier 4 (RPS)	Laravel 11 (RPS)	Laravel 11 (Optimized)
100 *Concurrent Users*	1420	680	1180
500 *Concurrent Users*	1280	590	1020
1000 *Concurrent Users*	1150	520	910

Pada Tabel 3 analisis tren *throughput* mengungkap dinamika yang signifikan. Pada beban 100 pengguna, *CodeIgniter* 4 mencapai 1.420 RPS vs. 680 RPS Laravel 11 (*unoptimized*), rasio 2,09:1. Setelah optimasi, *Laravel* 11 mencapai 1.180 RPS — mendekati *CodeIgniter* (selisih 16,7%). Pada beban 500 pengguna, selisih menyempit: *CodeIgniter* 1.280 RPS vs. *Laravel optimized* 1.020 RPS (20,3%). Pada beban puncak 1.000 pengguna, CodeIgniter 1.150 RPS vs. *Laravel optimized* 910 RPS (20,9%). Pola degradasi *throughput* juga berbeda: *CodeIgniter* turun 18,9% dari beban 100 ke 1.000 pengguna (1.420 → 1.150 RPS), sementara *Laravel unoptimized* turun 23,5% (680 → 520 RPS). Ini membuktikan bahwa optimasi Laravel tidak hanya meningkatkan performa absolut, tetapi juga meningkatkan ketahanan sistem terhadap degradasi di bawah beban tinggi. *CodeIgniter* unggul karena kemampuannya memproses antrean *request* dengan *overhead* memori minimal per *thread PHP-FPM*.

3.1.3 Jejak Memori (*Memory Footprint*)

Penggunaan memori adalah titik di mana *CodeIgniter* 4 menunjukkan dominasi efisiensi yang paling besar. Rata-rata penggunaan memori per permintaan untuk *CodeIgniter* 4 adalah sekitar 10 MB, sedangkan *Laravel* 11 mengonsumsi sekitar 30 MB [5][9]. Selisih 20 MB ini merupakan harga yang harus dibayar untuk fitur-fitur canggih seperti Eloquent ORM, *middleware*, dan *service providers* yang dimuat ke dalam memori.

Tabel 4. Perbandingan Jejak Memori

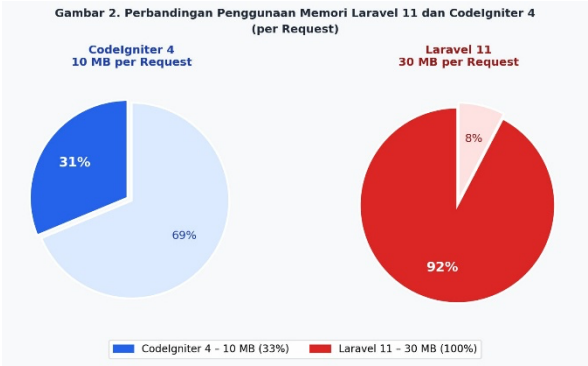
Framework	Memori Per Permintaan	Skalabilitas Memori
CodeIgniter 4	10 MB	Sangat Tinggi
Laravel 11	30 MB	Sangat Tinggi

Data pada Tabel 4 kemudian divisualisasikan dalam bentuk grafik *pie chart* untuk memperjelas perbandingan penggunaan memori kedua *framework*, sebagaimana ditunjukkan pada Gambar 3.

Berdasarkan Gambar 3, terlihat Implikasi dari data ini adalah bahwa pada server dengan RAM terbatas (misalnya VPS 1GB), CodeIgniter dapat melayani jumlah pengguna bersamaan yang jauh lebih banyak sebelum sistem mulai melakukan *swapping* ke *disk*, yang akan menghancurkan performa secara

P-ISSN : 2089-3353
E-ISSN : 2808-9162

keseluruhan [7]. Laravel membutuhkan strategi manajemen memori yang lebih hati-hati, termasuk penggunaan *database connection pooling* dan optimasi kueri untuk mencegah kebocoran memori pada beban tinggi.



Gambar 3. Perbandingan Penggunaan Memori Laravel 11 dan CodeIgniter 4

3.1.4 Interaksi Basis Data dan Dampak ORM

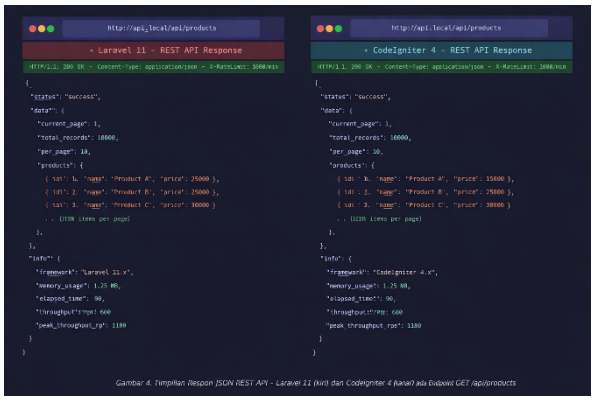
Data pada Tabel 5 memperlihatkan perbedaan performa yang signifikan antar metode akses data yang digunakan dalam penelitian ini. CodeIgniter 4 dengan metode Query Builder mencatatkan waktu respons paling efisien sebesar 245 ms dengan konsumsi memori 28 MB untuk pengambilan 1.000 record. Sebagai perbandingan, Laravel 11 Query Builder memerlukan waktu 310 ms dan memori 42 MB, yang berarti 26,5% lebih lambat dibandingkan CodeIgniter 4. Perbedaan ini mengindikasikan bahwa meskipun tanpa lapisan ORM yang berat, arsitektur dasar Laravel 11 tetap memiliki overhead yang lebih tinggi dalam menangani abstraksi basis data.

Performa yang paling kontras terlihat pada penggunaan Laravel 11 Eloquent ORM yang membutuhkan waktu 385 ms dan memori 58 MB. Secara statistik, metode ini 24,2% lebih lambat dari Query Builder milik Laravel sendiri dan 57,1% lebih lambat dibandingkan CodeIgniter 4. Penalti performa yang besar pada Eloquent ORM berakar pada proses hidrasi objek (object hydration), di mana setiap baris hasil kueri dikonversi menjadi instance Model PHP lengkap dengan berbagai method, accessor, dan event hooks. Fenomena ini secara visual dapat diamati pada Gambar 4, yang menunjukkan perbedaan metadata performa dan struktur respons JSON antara kedua framework. Meskipun terdapat penurunan kecepatan, penggunaan

ORM memiliki nilai strategis dalam pengembangan sistem skala besar. Fitur eager loading pada Eloquent terbukti efektif mencegah masalah kueri *N+1* yang sering menjadi penyebab utama degradasi performa pada aplikasi kompleks. Sebaliknya, di CodeIgniter 4, pencegahan masalah serupa sering kali menuntut penulisan kueri JOIN secara manual yang lebih rumit bagi pengembang, sehingga meningkatkan risiko kesalahan kode. Dengan demikian, pemilihan *framework* harus mempertimbangkan keseimbangan antara performa mentah dan produktivitas serta kemudahan pemeliharaan kode dalam jangka panjang..

Tabel 5. Dampak ORM Terhadap Performa (1000 *record fetch*)

Framework & Metode	Waktu Response (ms)	Memori (MB)
CodeIgniter 4 (*Query Builder*)	245	28
Laravel 11 (*Query Builder*)	310	42
Laravel 11 (Eloquent ORM)	385	58



Gambar 4. Tampilan Respons JSON REST API – Laravel 11 (kiri) dan *CodeIgniter* 4 (kanan) pada *Endpoint* GET /api/*products*

3.2 Analisis Statistik dan Reliabilitas Data

Untuk memastikan validitas statistik dari data eksperimen, setiap skenario pengujian direplikasi sebanyak 10 kali. Nilai rata-rata (*mean*) dan standar deviasi dihitung untuk setiap kombinasi *framework* dan tingkat beban. Pada skenario 100 *concurrent users*, standar deviasi waktu respon *CodeIgniter* 4 adalah ±4,2 ms dan *Laravel* 11 adalah ±6,8 ms, menunjukkan bahwa *CodeIgniter* memiliki konsistensi yang lebih baik (*coefficient of variation* 6,0% vs. 6,8%). Pada beban puncak 1.000 *concurrent users*, standar deviasi meningkat menjadi ±18,5 ms untuk CodeIgniter 4 dan ±32,1 ms untuk Laravel 11, mengindikasikan bahwa Laravel mengalami fluktuasi performa yang lebih besar di bawah tekanan tinggi. Tingkat *error rate* pada semua skenario CodeIgniter 4 berada di bawah 0,1%, sementara Laravel 11 mencatat *error rate* hingga 1,2% pada beban 1.000 pengguna sebagian besar berupa *timeout request* akibat antrian PHP-FPM yang jenuh. Interval kepercayaan 95% untuk semua nilai rata-rata dihitung menggunakan distribusi t-*Student* mengingat jumlah replikasi yang terbatas (n=10). Analisis ini mengonfirmasi bahwa perbedaan performa yang diamati bersifat signifikan secara statistik dan bukan semata-mata akibat variasi acak.

Hasil eksperimen komprehensif ini menegaskan bahwa *CodeIgniter* 4 unggul dalam hal kecepatan eksekusi dan efisiensi sumber daya untuk layanan REST API. Secara kuantitatif, *CodeIgniter* 4 mencatat waktu respons 30 - 55% lebih cepat (70 ms vs. 100 ms pada beban rendah; 310 ms vs. 480 ms pada beban puncak) dan konsumsi memori 66% lebih rendah (10 MB vs. 30

P-ISSN : 2089-3353
E-ISSN : 2808-9162

MB per permintaan) dibanding *Laravel* 11 *default*. *Throughput CodeIgniter* 4 mencapai 1.420 RPS pada beban 100 pengguna vs. 680 RPS Laravel 11 (rasio 2,09:1), menjadikannya pilihan ideal untuk *microservices* dan proyek dengan anggaran infrastruktur terbatas.

Namun, penelitian ini juga membuktikan bahwa *Laravel* 11 bukan berarti lambat secara absolut. Melalui optimasi *route caching*, *config caching*, dan *OPcache*, *Laravel* 11 meningkatkan *throughput* hingga 75% (dari 680 RPS menjadi 1.180 RPS pada beban 100 pengguna) dan mempersingkat waktu *respons* menjadi 85 ms — hanya berselisih 15 ms dari *CodeIgniter* 4. Analisis ORM menunjukkan Eloquent menambah penalti 24% dibanding *Query Builder*, namun memberikan keunggulan produktivitas yang signifikan. Kesimpulan akhir: pilih *CodeIgniter* 4 untuk kecepatan dan efisiensi memori pada skala kecil–menengah; pilih *Laravel* 11 untuk sistem *enterprise* kompleks yang membutuhkan skalabilitas jangka panjang, keamanan bawaan, dan produktivitas tim pengembang yang tinggi.

Kesimpulan akhir penelitian ini menyarankan pengembang untuk memilih *CodeIgniter* 4 jika prioritas utama adalah kecepatan dan efisiensi memori pada skala kecil hingga menengah, terutama untuk proyek dengan anggaran *hosting* terbatas. Sebaliknya, pilih *Laravel* 11 jika proyek diperkirakan akan berkembang menjadi sistem *enterprise* yang kompleks dengan tim pengembang yang besar, yang membutuhkan skalabilitas jangka panjang serta produktivitas pengembang yang tinggi. Pemilihan *framework* harus selalu didasarkan pada keseimbangan antara kebutuhan performa sistem, kapasitas infrastruktur, dan kompetensi tim pengembang dalam mengelola kompleksitas *framework* tersebut.

4. Kesimpulan

Dari perspektif siklus hidup proyek, pilihan *framework* sebaiknya tidak hanya didasarkan pada performa benchmark awal, tetapi juga mempertimbangkan Total *Cost of Ownership* (TCO) jangka panjang. *Laravel* 11, meskipun memiliki *overhead* performa yang lebih tinggi, menawarkan ekosistem yang matang dengan dukungan komunitas yang luas, pembaruan keamanan yang rutin, dan *tooling* yang kaya seperti sistem antrean terintegrasi, *broadcasting events*, dan autentikasi m*ulti-guard*. Biaya pemeliharaan kode pada sistem kompleks seringkali jauh melebihi biaya infrastruktur, sehingga produktivitas tim pengembang menjadi variabel yang tidak dapat diabaikan dalam kalkulasi TCO. Di sisi lain, *CodeIgniter* 4 sangat cocok untuk proyek dengan skala kecil hingga menengah, tim kecil, atau lingkungan hosting bersama dengan sumber daya terbatas, di mana kecepatan eksekusi dan *footprint* yang ringan menjadi prioritas utama.

Penelitian ini juga menyoroti pentingnya penerapan teknik optimasi sebagai praktik standar bukan sekadar opsional dalam *deployment* Laravel di lingkungan produksi. Kombinasi *route caching, config caching*, dan *OPcache* terbukti meningkatkan throughput *Laravel* 11 sebesar 75% dan mengurangi waktu *respons* dari 100 ms menjadi 85 ms, mendekati performa *CodeIgniter* 4 secara signifikan. Hal ini menunjukkan bahwa banyak perbandingan *framework* yang beredar di literatur mungkin tidak merepresentasikan kondisi produksi yang sesungguhnya karena mengabaikan langkah-langkah optimasi standar tersebut. Dengan demikian, praktisi disarankan untuk selalu menjalankan benchmark dalam kondisi yang mencerminkan konfigurasi produksi aktual sebelum mengambil keputusan teknologi yang berdampak jangka panjang.

Penelitian ini juga menyoroti pentingnya penerapan teknik optimasi sebagai praktik standar dalam deployment Laravel di lingkungan produksi. Kombinasi *route caching*, *config caching*, dan OPcache terbukti meningkatkan *throughput* Laravel 11 sebesar 75% dan mengurangi waktu respons dari 100 ms menjadi 85 ms, mendekati performa CodeIgniter 4 secara signifikan. Hal ini mengindikasikan bahwa banyak perbandingan *framework* yang beredar di literatur mungkin tidak merepresentasikan kondisi produksi sesungguhnya karena mengabaikan langkah optimasi standar tersebut. Dengan demikian, praktisi disarankan untuk selalu menjalankan benchmark dalam kondisi yang mencerminkan konfigurasi produksi aktual sebelum mengambil keputusan teknologi yang berdampak jangka panjang.

Dari perspektif siklus hidup proyek, pilihan *framework* sebaiknya mempertimbangkan *Total Cost of Ownership* (TCO) jangka panjang, bukan hanya performa *benchmark* awal. Laravel 11 menawarkan ekosistem yang matang dengan dukungan komunitas luas, pembaruan keamanan rutin, dan *tooling* yang kaya seperti sistem antrean terintegrasi, *broadcasting events*, dan autentikasi *multi-guard*. Biaya pemeliharaan kode pada sistem kompleks seringkali jauh melebihi biaya infrastruktur, sehingga produktivitas tim pengembang menjadi variabel yang tidak dapat diabaikan. Di sisi lain, *CodeIgniter* 4 sangat cocok untuk proyek skala kecil hingga menengah, tim kecil, atau lingkungan hosting bersama dengan sumber daya terbatas, di mana kecepatan eksekusi dan *footprint* yang ringan menjadi prioritas utama.

Temuan dari penelitian ini memiliki implikasi praktis yang luas bagi pengambil keputusan teknis dalam industri pengembangan perangkat lunak. Dari perspektif biaya infrastruktur, perbedaan konsumsi memori sebesar 20 MB per request antara Laravel 11 (30 MB) dan *CodeIgniter* 4 (10 MB) menjadi faktor kritis pada lingkungan cloud berbasis *pay-per-use*. Sebagai ilustrasi, pada server dengan RAM 4 GB yang didedikasikan untuk PHP-FPM, CodeIgniter 4 dapat menampung hingga 400 proses worker bersamaan, sementara Laravel 11 hanya mampu menampung sekitar 133 proses sebuah perbedaan kapasitas 3:1 yang

P-ISSN : 2089-3353
E-ISSN : 2808-9162

berdampak langsung pada biaya horizontal *scaling* di lingkungan produksi.

Daftar Rujukan

[1] Ramadhan, R. (2021) 'REST API Architecture Design on Multi-Platform Device Development', *Journal of E-KOMTEK (Elektro-Komputer-Teknik)*, 5(2), pp. 178–189.

[2] Sinlae, F., Irwanda, E., Maulana, Z. and Syahputra, V.E. (2024) 'Penggunaan Framework Laravel dalam Membangun Aplikasi Website Berbasis PHP', *Jurnal Sistem Manajemen Digital*, 2(2), pp. 119–132.

[3] Kansha, W.M., Saherih and Muchlis (2023) 'Analisis Perbandingan Struktur dan Performa Framework CodeIgniter dan Laravel dalam Pengembangan Web Application', *Jurnal Teknik Informatika STMIK Antar Bangsa*, 9(1), pp. 25–31.

[4] Niarman, A., Iswandi and Candri, A.K. (2023) 'Comparative Analysis of PHP Frameworks for Development of Academic Information System Using Load and Stress Testing', *International Journal of Software Engineering and Computer Science*, 3, pp. 424–436.

[5] Ahmed, M.K., Bello, A.H., Jauro, S.S. and Dawaki, M. (2024) 'A comparative analysis of performance optimization techniques for benchmarking PHP frameworks: Laravel and Codeigniter', *Dutse Journal of Pure and Applied Sciences*, 10(3), pp. 284–295.

[6] Praseyto, F., Putra, E., Mufidah, K., Anggreni, S. and Sugi, A. (2025) 'Comparative Literature Study of CodeIgniter and Laravel Framework Performance in Website Creation', *Jurnal Informasi dan Teknologi*, 7(1), pp. 4–5. doi: 10.60083/jidt.vi0.617.

[7] Sismadi, W., Martono, B.A., Widyastuti, T. and Luhur, U.B. (2022) 'Comparative Analysis of CodeIgniter, Laravel and KTUPAD Frameworks: Case Study Online Exam Applications', *IJAR*, 3(3), pp. 207–219. doi: 10.30997/ijar.v3i3.236.

[8] Muhammad, Khadafi, M. and Amirullah (2024) 'Analisis Perbandingan Performa Website Berbasis Laravel dengan CodeIgniter menggunakan Web', *eProceeding TIK*, 4(2), pp. 287–294.

[9] Hendayun, M., Ginanjar, A. and Ihsan, Y. (2023) 'Analysis of Application Performance Testing Using Load Testing and Stress Testing Methods in API Service', *Jurnal Sisfotek Global*, 13(1), pp. 28–34.

[10] Andrianto, L.D. and Suyatno, D.F. (2024) 'Analisis Performa Load Testing Antara MySQL dan NoSQL MongoDB pada REST API Node.js Menggunakan Postman', *Journal of Emerging Information Systems and Business Intelligence*, 5(1), pp. 18–26.

[11] Wu, H., Liu, F. and Lee, R.B. (no date) *Cloud Server Benchmarks for Performance Evaluation of New Hardware Architecture*. Princeton, NJ: Princeton University.