Department of Digital Business
Journal of Artificial Intelligence and Digital Business (RIGGS)
Homepage: https://journal.ilmudata.co.id/index.php/RIGGS
Vol. 5 No. 1 (2026) pp: 12113-12119
P-ISSN: 2963-9298, e-ISSN: 2963-914X

Perbandingan Performa RESTful API Menggunakan Framework Python Flask dan Go Gin pada Pengembangan Aplikasi Alumni Circle

I Kadek Dwi Angga Sathya Nanda, Ketut Agus Seputra, Kadek Teguh Dermawan
Program Studi S1 Sistem Informasi, Jurusan Teknik Informatika, Fakultas Teknik dan Kejuruan, Universitas Pendidikan Ganesha, Singaraja, Bali, Indonesia
dwi.angga@student.undiksha.ac.id, agus.seputra@undiksha.ac.id, kdermawan@undiksha.ac.id

Abstrak

Pemilihan framework backend yang tepat merupakan keputusan krusial dalam pengembangan aplikasi web, terutama ketika sistem dituntut untuk melayani banyak pengguna secara bersamaan dengan sumber daya komputasi yang terbatas. Penelitian ini membandingkan performa Python Flask dan Go Gin yang diimplementasikan pada sistem Alumni Circle melalui pengujian beban (load testing) dengan skenario mixed workload sebanyak 500 pengguna simultan selama 5 menit. Aspek yang diukur meliputi throughput (Requests Per Second), latensi, serta penggunaan CPU dan memori yang dipantau secara real-time menggunakan docker stats. Hasil pengujian menunjukkan bahwa Go Gin memproses 55.739 permintaan dengan tingkat keberhasilan 100% dan rata-rata 185,88 RPS, sementara Python Flask hanya menyelesaikan 29.201 permintaan dengan 7.322 kegagalan dan rata-rata 97,36 RPS. Dari sisi latensi, Go Gin mencatat rata-rata waktu respons 468,18 ms, hampir lima kali lebih cepat dibanding Flask yang mencapai 2.332,72 ms. Pada kondisi beban puncak, Flask mencatatkan latensi maksimal hingga 132.860 ms, sedangkan Go Gin mampu meredam lonjakan di angka 43.117 ms tanpa menghasilkan error. Dalam hal sumber daya, rata-rata penggunaan CPU kedua framework relatif sebanding, namun Go Gin lebih stabil di beban puncak dengan CPU tertinggi 41,21% dibanding Flask 60,82%. Efisiensi memori Go Gin juga lebih unggul dengan rata-rata 53,54 MB berbanding 114,65 MB pada Flask. Berdasarkan hasil tersebut, Go Gin terbukti lebih unggul dalam throughput, stabilitas latensi, dan efisiensi memori dibanding Python Flask pada skenario beban tinggi. Go Gin direkomendasikan sebagai arsitektur backend yang lebih andal dan skalabel untuk aplikasi Alumni Circle maupun sistem sejenis yang mengutamakan performa dan efisiensi sumber daya server.

Kata kunci: Go Gin, Python Flask, Load Testing, Latensi, Throughput, Sumber Daya Server

1. Latar Belakang

Dalam rangka menangani peningkatan jumlah pengguna yang terhubung ke internet, teknologi server side terus dikembangkan tidak hanya untuk meningkatkan kemudahan pembuatan aplikasi, tetapi juga untuk mengoptimalkan performa sistem. Kecepatan dan efisiensi aplikasi server menjadi aspek yang sangat krusial, di mana aplikasi yang ideal adalah aplikasi yang mampu menangani volume permintaan (request) yang besar dari client tanpa mengonsumsi sumber daya komputasi secara berlebihan. Optimalisasi penggunaan sumber daya ini secara langsung berdampak pada penekanan biaya operasional infrastruktur yang harus dikeluarkan oleh pengelola layanan.

Salah satu sistem yang memerlukan performa server yang handal adalah sistem informasi alumni. Tracer study dan pengelolaan data alumni merupakan kebutuhan penting bagi institusi pendidikan, baik sebagai alat evaluasi kinerja lulusan maupun sebagai salah satu syarat kelengkapan akreditasi [1]. Penelitian di Politeknik Manufaktur Bandung menunjukkan bahwa data alumni yang terkelola dengan baik mampu menghasilkan informasi strategis mengenai profil lulusan, mulai dari masa tunggu kerja hingga kesesuaian kompetensi dengan dunia industri [2]. Lebih lanjut, implementasi sistem informasi manajemen alumni berbasis web juga terbukti meningkatkan keterlibatan alumni secara signifikan, memperluas akses informasi beasiswa dan peluang kerja, serta mempererat hubungan antara alumni dan institusi pendidikan [3]. Kebutuhan-kebutuhan tersebut mendorong pengembangan sistem Alumni Circle sebagai platform terpadu yang melayani berbagai fitur mulai dari pengelolaan data hingga unggah berkas, yang pada gilirannya menuntut infrastruktur backend yang mampu menangani beban tinggi secara efisien.

Performa backend sebuah sistem tidak hanya ditentukan oleh desain aplikasinya, tetapi juga oleh efisiensi pengelolaan data dan kemampuan infrastruktur dalam menangani lonjakan trafik. Penelitian mengenai optimasi query database menunjukkan bahwa penerapan teknik indexing yang tepat mampu mengurangi waktu eksekusi hingga 90,20%, yang membuktikan betapa pentingnya perancangan sistem yang berorientasi pada performa sejak tahap awal [4]. Hal ini sejalan dengan temuan pada pengujian beban (load testing) website perpustakaan Universitas Malikussaleh menggunakan Apache JMeter, yang mengungkapkan bahwa sistem mulai mengalami degradasi performa secara signifikan ketika jumlah pengguna simultan melampaui kapasitas infrastruktur yang tersedia [5]. Kedua penelitian tersebut menegaskan bahwa pengujian performa secara terukur merupakan langkah yang tidak dapat diabaikan dalam pembangunan sistem yang ditujukan untuk banyak pengguna.

Dalam konteks pemilihan framework backend, sejumlah penelitian telah membandingkan berbagai teknologi dari sisi latensi, throughput, dan penggunaan sumber daya. Perbandingan antara Laravel Octane dan Webman menunjukkan bahwa arsitektur non-blocking event-driven mampu menghasilkan throughput hampir dua kali lipat dengan latensi yang jauh lebih rendah, membuktikan bahwa pilihan arsitektur framework berpengaruh langsung terhadap efisiensi sistem [6]. Temuan serupa juga ditemukan pada perbandingan NestJS dan Lumen, di mana meskipun perbedaan waktu respons antar keduanya relatif kecil, NestJS menunjukkan efisiensi yang lebih baik dalam penggunaan CPU dan memori [7]. Penelitian-penelitian tersebut memberikan landasan empiris bahwa pemilihan framework yang tepat harus mempertimbangkan kebutuhan performa aktual, bukan sekadar popularitas atau kemudahan pengembangan semata.

Saat ini, pemilihan bahasa pemrograman dan framework menjadi tantangan tersendiri dalam pengembangan layanan berbasis Representational State Transfer (REST). Python Flask telah lama menjadi pilihan populer karena fleksibilitasnya, namun di sisi lain, Go (Golang) dengan framework Gin muncul sebagai kompetitor kuat yang menjanjikan performa tinggi melalui efisiensi kompilasi dan manajemen concurrency. Penelitian ini bertujuan untuk mengukur serta membandingkan performa antara Python Flask dan Go Gin yang diimplementasikan pada sistem Alumni Circle. Hasil penelitian ini diharapkan dapat menjadi panduan bagi pengambil keputusan dalam menentukan arsitektur backend yang paling tepat untuk membangun aplikasi web yang responsif dan efisien.

Aspek performa yang diukur dalam pengujian ini meliputi penggunaan CPU, penggunaan RAM, serta kecepatan respons (latency dan throughput) melalui metode load testing dengan beban campuran (mixed workload). Faktor lain seperti keamanan, kemudahan pengembangan, serta performa pada sisi client berada di luar batasan penelitian ini. Fokus utama tetap tertuju pada bagaimana setiap framework mengelola sumber daya server saat dihadapkan pada skenario penggunaan dunia nyata, seperti pengolahan data alumni hingga proses unggahan file pada sistem.

Beberapa penelitian terdahulu telah memberikan kontribusi signifikan dalam domain evaluasi performa backend. Studi fundamental oleh [8] mengawali perbandingan komputasi pada algoritma numerik yang menunjukkan bahwa bahasa bertipe *compiled* seperti Go memiliki keunggulan kecepatan eksekusi yang signifikan dibandingkan bahasa interpreted seperti Python saat menangani beban kerja besar. Di sisi lain, evaluasi terhadap berbagai framework populer mengungkapkan bahwa meskipun Flask memiliki skalabilitas yang baik, framework Go Gin dinilai sangat sesuai untuk pemrosesan data skala menengah dengan performa yang lebih stabil [9]. Efisiensi ini juga terlihat pada level akses data, di mana penggunaan *Object Relational Mapping* (ORM) berbasis Go seperti GORM terbukti jauh lebih cepat dalam menangani operasi CRUD dibandingkan SQLAlchemy pada Python [10].

Analisis komparatif lainnya menunjukkan bahwa kombinasi Go dengan basis data relasional (MySQL) lebih unggul dalam efisiensi penggunaan CPU dan memori dibandingkan Node.js [11], serta menunjukkan dominasi pada protokol komunikasi real time dengan throughput yang mencapai lebih dari 44.000 pesan per detik pada ribuan klien konkuren [12]. Namun, penelitian ini secara spesifik berupaya mengisi celah literatur dengan melakukan analisis komparatif efisiensi antara Python Flask dan Go Gin pada implementasi sistem informasi alumni yang melibatkan karakteristik beban kerja I/O serta proses komputasi tertentu.

2. Metode Penelitian

2.1. Lingkungan Percobaan

Penelitian ini dilakukan dengan menggunakan dua unit komputer, kedua komputer tersebut memiliki peranan masing-masing. Komputer yang pertama berperan sebagai server yang didalamnya terpasang semua aplikasi obyek penelitian, sementara komputer kedua bertindak sebagai *client* yang memiliki tujuan untuk mengirimkan *request*

DOI: https://doi.org/10.31004/riggs.v5i1.7664

kepada komputer *server*. Hal ini dilakukan dengan tujuan agar memisahkan konteks kerja antara sisi *client* dan *server* supaya tidak tumpang tindih, komputer yang berperan sebagai sisi *client* akan berfokus pada proses *request* dan tidak mempengaruhi kinerja dari aplikasi-aplikasi yang akan diuji pada komputer yang berperan manjadi *server*. Kedua komputer tersebut akan dihubungkan melalui IP *Address* lokal *wifi* yang sama agar keduanya bisa saling berkomunikasi.

Komputer yang digunakan dalam penelitian ini memiliki spesifikasi yang berbeda satu sama lain, komputer yang akan dijadikan *client* memiliki spesifikasi CPU 1.3 GHz Intel Core i5, RAM 4GB, serta sistem operasi macOS Mojave 10.14.6. Sementara komputer yang akan dijadikan server memiliki spesifikasi CPU Apple M4, RAM 16GB, serta sistem operasi macOS Tahoe 26.3.1.

2.2. Aplikasi Obyek Penelitian

Aplikasi yang akan dijadikan obyek penelitian merupakan aplikasi berbasis *mobile* dengan nama Alumni Circle, aplikasi ini dirancang untuk mengintegrasikan data lulusan, informasi lowongan pekerjaan, dan agenda kegiatan alumni dalam satu kesatuan sistem. Pengguna aplikasi diwajibkan melakukan proses login terlebih dahulu sebelum bisa mengakses informasi yang disediakan oleh aplikasi tersebut. Otentikasi pengguna dilakukan dengan metode username dan password yang telah terdaftar pada aplikasi, password disimpan dalam basis data bentuk hash dengan algoritma hash Bcrypt. Pengguna yang berhasil melakukan proses login akan diberikan sebuah token yang dipakai untuk mengakses informasi-informasi terbatas atau dilindungi pada aplikasi, token ini disusun menggunakan JSON Web Token (JWT). Token JWT yang didapatkan saat berhasil login harus disertakan dalam *Authorization request header* bertipe *Bearer* yang dikirimkan oleh *client* agar dapat memperoleh informasi terbatas atau dilindungi yang mewajibkan otentikasi.

Aplikasi ini menyediakan informasi berdasarkan endpoint yang diminta oleh *client*, informasi yang sukses ke sisi client akan disajikan dalam format *JavaScript Object Notation* (JSON). Tabel satu menunjukkan endpoint yang disediakan oleh aplikasi dan bisa diakses oleh *client* untuk mendapatkan informasi-informasi yang diperlukan, beserta keterangan-keterangan lain.

Tabel 1. Daftar Endpoint

Endpoint	Method	Tujuan	Keterangan
/api/v1/private/vacancy	POST	Menambah daftar lowongan pekerjaan.	Menguji performa service dalam menangani proses write yang intensif, khususnya pada pemrosesan *multipart form-data* saat mengunggah file gambar.
/api/v1/private/event	GET	Mendapatkan data daftar kegiatan.	Menguji efisiensi service dalam manangani proses logika bisnis yang kompleks, seperti proses *join operation* antar tabel.
/api/v1/private/alumni	GET	Mendapatkan data daftar alumni.	Menguji kecepatan response dasar untuk data yang *plain select*.

Perangkat sistem pengelolaan basis data yang digunakan dalam pengembangan kedua aplikasi tersebut adalah MySQL versi 9.6.0.

Aplikasi RESTful Python akan dibangun di atas Python versi 3.12.0, framework yang digunakan untuk membangun aplikasi ini adalah Flask versi 3.0.0. Koneksi basis data pada aplikasi ini menggunkan library mysql-

DOI: https://doi.org/10.31004/riggs.v5i1.7664

connector-python versi 8.4.0 sebagai driver MySQL. Serta implementasi otentikasi dan otorisasi dengan JWT dan Bcrypt menggunakan library flask-jwt-extended versi 4.6.0 dan flask-bcrypt versi 1.0.1.

Aplikasi RESTful Go akan dibangun di atas Go versi 1.24, framework yang digunakan untuk membangun aplikasi ini adalah Gin versi 1.10.1. Koneksi basis data pada aplikasi ini menggunkan library gorm.io/driver/mysql versi 1.6.0 sebagai driver MySQL. Serta implementasi otentikasi dan otorisasi dengan JWT dan Bcrypt menggunakan library github.com/golang-jwt/jwt versi 3.2.2 dan golang.org/x/crypto/bcrypt versi 0.49.0.

2.3. Perangkat Pengambilan Data

Pengukuran performa kecepatan respons dan ketahanan aplikasi akan dilakukan menggunakan perangkat lunak Locust. Locust adalah alat pengujian beban (load testing) berbasis Python yang memungkinkan simulasi ribuan pengguna secara serentak untuk mengukur perilaku sistem di bawah tekanan. Penggunaan Locust sebagai instrumen pengujian beban dalam penelitian ini didasarkan pada kemampuannya dalam mensimulasikan pengguna secara serentak sekaligus mengukur indikator kinerja utama seperti throughput, response time, dan error rate pada sistem berbasis web [13]. Dalam penelitian ini, Locust dikonfigurasi menggunakan skrip khusus untuk menjalankan beban campuran (mixed workload) yang merepresentasikan aktivitas nyata pada aplikasi Alumni Circle. Pengujian dilakukan dengan skenario 500 pengguna virtual (concurrent users) dengan durasi pengujian selama 5 menit untuk mendapatkan data throughput (RPS) serta latensi yang stabil.

Parameter latensi yang diukur meliputi nilai rata-rata (Average) dan persentil ke-95 (95th Percentile). Pengukuran response time pada HTTP request merupakan metode dasar yang paling umum digunakan untuk mengevaluasi kualitas layanan web, di mana akurasi pengukurannya sangat bergantung pada metode dan kondisi pengambilan data yang digunakan [14]. Oleh karena itu, selain nilai rata-rata, pengukuran pada persentil ke-95 turut disertakan untuk mencerminkan pengalaman pengguna pada kondisi beban yang sesungguhnya, bukan hanya kondisi ideal.

Untuk pengukuran konsumsi sumber daya CPU dan memori (RAM), penelitian ini menggunakan fitur pemantauan real-time melalui perintah docker stats. Pendekatan pemantauan berbasis container ini memungkinkan pengambilan data penggunaan CPU, memori, dan sumber daya lainnya secara terintegrasi dalam satu lingkungan yang terisolasi, sehingga hasil pengukuran tidak tercampur dengan proses lain di luar kontainer yang diuji [15]. Pengambilan data dilakukan secara semi otomatis dengan menyalurkan (streaming) output dari docker stats ke dalam sebuah file berformat CSV selama pengujian berlangsung. Data tersebut kemudian diolah untuk mendapatkan nilai penggunaan puncak (peak usage) serta nilai rata-rata (average) dari masing-masing container (Flask dan Gin).

2.4. Prosedur Pengambilan Data

Prosedur pengumpulan data dilakukan secara sistematis dengan tahapan sebagai berikut:

1. Menjalankan layanan backend (Flask atau Gin) dalam container Docker hingga status ready.
2. Membuka terminal docker stats untuk memantau penggunaan sumber daya awal (idle).
3. Menjalankan Locust dengan konfigurasi 500 *users* dan *spawn rate* 10 *users/s* selama 5 menit menggunakan skenario *mixed task.*
4. Melakukan semi otomatis data CPU dan RAM dari docker stats dengan streaming output ke dalam file CSV.
5. Menghentikan container, membersihkan cache Docker (pruning), dan memulai ulang prosedur untuk framework berikutnya guna menjamin validitas data.

3. Hasil dan Diskusi

3.1 *Troughput* (RPS)

Pengujian beban yang dilakukan selama 5 menit dengan kapasitas 500 pengguna dan kecepatan peningkatan 10 pengguna per detik menunjukkan perbedaan kemampuan yang cukup besar antara kedua sistem. Go Gin berhasil menyelesaikan total 55.739 permintaan tanpa ada satu pun yang gagal. Di sisi lain, Python Flask hanya mampu memproses 29.201 permintaan, dan dari jumlah tersebut terdapat 7.322 permintaan yang gagal. Hal ini membuat

DOI: https://doi.org/10.31004/riggs.v5i1.7664

tingkat keberhasilan Go Gin mencapai 100%, sedangkan Python Flask hanya sebesar 74,92%. Tabel 2 berikut akan merinci perbandingan jumlah permintaan dan RPS yang dihasilkan.

Tabel 2. Hasil Perbandingan Throughput

Parameter Pengujian	Python Flask	Go Gin
Total Permintaan	29.201	55.739
Permintaan Berhasil	21.879	55.739
Permintaan Gagal	7.322	0
Tingkat Keberhasilan (%)	74,92%	100%
Rata-rata RPS	97,36	185,88

Dari sisi kecepatan pengiriman data (RPS), Go Gin mampu bekerja hampir dua kali lipat lebih cepat dengan rata-rata 185,88 RPS dibandingkan Python Flask yang hanya 97,36 RPS. Perbedaan ini sangat terlihat pada endpoint /alumni dan /event, di mana Go Gin tetap stabil di angka 36-37 RPS, sementara Flask turun drastis hingga di bawah 1 RPS. Bahkan pada proses berat seperti unggah gambar di /vacancy, Go Gin masih unggul jauh dengan 111,64 RPS. Data ini menunjukkan bahwa Go Gin jauh lebih tangguh dalam menangani banyak pengguna sekaligus tanpa merusak sistem.

3.2 Latensi

Hasil pengukuran latensi selama durasi 5 menit mengonfirmasi adanya hambatan pemrosesan yang berat pada sisi Python Flask dibandingkan efisiensi yang ditunjukkan oleh Go Gin. Rata-rata waktu respon (Average Response Time) Go Gin tercatat sebesar 468,18 ms, yang secara teknis 4,9 kali lebih cepat dibandingkan rata-rata respon Python Flask yang menyentuh angka 2.332,72 ms. Perbedaan durasi hingga 2,3 detik pada Flask ini menunjukkan terjadinya antrean panjang di tingkat backend yang berpotensi merusak pengalaman pengguna secara sistemik. Bahkan pada kondisi beban paling ringan (Minimum Response Time), Go Gin tetap lebih unggul dengan kecepatan 9 ms dibanding Flask yang memerlukan waktu minimal 20 ms. Tabel 3 berikut akan merinci perbandingan kecepatan respon yang dihasilkan.

Tabel 3. Hasil Perbandingan Kecepatan Respon

Parameter Pengujian	Python Flask	Go Gin
Minimum Response Time	20 ms	9 ms
Average Response Time	2.332,72 ms	468,18 ms
Maximum Response Time	132.860 ms	43.117 ms

Kesenjangan performa mencapai puncaknya pada metrik latensi maksimal (Maximum Response Time). Python Flask mencatatkan waktu tunggu ekstrem hingga 132.860 ms atau sekitar 2,2 menit. Kondisi ini menandakan sistem hampir membeku (unresponsive) saat menangani beban campuran. Di sisi lain, Go Gin mampu meredam lonjakan latensi maksimal di angka 43.117 ms. Meskipun angka maksimal tersebut dipicu oleh beratnya beban 500 pengguna pada proses unggah gambar, Go Gin tetap mampu menyelesaikan permintaan tersebut tanpa menghasilkan error. Hal ini berbeda dengan Flask yang sering kali memutus koneksi karena melampaui batas waktu tunggu (timeout). Secara keseluruhan, stabilitas latensi Go Gin membuktikan keunggulan manajemen memori dan concurrency dalam menjaga responsivitas sistem Alumni Circle.

DOI: https://doi.org/10.31004/riggs.v5i1.7664

3.3 Penggunaan Sumber Daya

Pengukuran penggunaan sumber daya dilakukan untuk mengetahui efisiensi tiap framework dalam memanfaatkan infrastruktur server saat berada di bawah beban 500 pengguna. Data diambil melalui fitur docker stats yang mencatat performa CPU dan memori secara real-time selama pengujian berlangsung. Hasil rata-rata dan nilai puncak penggunaan sumber daya dirangkum pada Tabel 4.

Tabel 4. Hasil Perbandingan Penggunaan Sumber Daya

Parameter Pengujian	Python Flask	Go Gin	Keterangan
Rata-rata Penggunaan CPU	24,81%	27,41%	Keduanya relatif setara.
Penggunaan CPU Tertinggi (Peak)	60,82%	41,21%	Go 1,5x lebih stabil
Rata-rata Penggunaan Memori	114,65 MB	53,54 MB	Go 2,1x lebih hemat
Penggunaan Memori Tertinggi	122,70 MB	62,08 MB	Go ~2x lebih hemat

Berdasarkan hasil pengujian, rata-rata penggunaan CPU kedua framework menunjukkan angka yang relatif sebanding, yaitu 24,81% untuk Python Flask dan 27,41% untuk Go Gin. Hal ini menunjukkan bahwa pada kondisi beban normal, kedua framework memanfaatkan sumber daya CPU secara setara. Perbedaan yang lebih terlihat justru muncul pada nilai puncak (peak) CPU, di mana Flask mencapai 60,82% sementara Go Gin hanya menyentuh 41,21%. Selisih ini menunjukkan bahwa Go Gin lebih konsisten dalam mengelola lonjakan beban, sehingga lebih kecil risikonya mengalami penurunan performa saat trafik meningkat secara tiba-tiba.

Dalam hal penggunaan memori, Go Gin menunjukkan keunggulan yang lebih signifikan. Rata-rata pemakaian memori Go Gin hanya sebesar 53,54 MB, sementara Python Flask membutuhkan 114,65 MB sekitar 2,1 kali lebih besar. Perbedaan ini mencerminkan karakteristik mendasar kedua teknologi: binary Go yang dikompilasi bersifat self-contained dan tidak memerlukan runtime tambahan, sedangkan interpreter Python harus memuat berbagai dependensi ke dalam memori sejak awal. Kondisi ini juga tercermin pada nilai puncak memori masing-masing, di mana Go Gin hanya mencapai 62,08 MB berbanding Flask yang menyentuh 122,70 MB.

Secara keseluruhan, data penggunaan sumber daya menunjukkan bahwa Go Gin lebih unggul terutama dalam efisiensi memori dan stabilitas CPU di bawah beban puncak. Keunggulan ini menjadikan Go Gin pilihan yang lebih ekonomis untuk skalabilitas aplikasi Alumni Circle, karena mampu melayani trafik yang sama dengan jejak memori yang lebih kecil pada spesifikasi perangkat keras yang setara.

4. Kesimpulan

Berdasarkan hasil pengujian beban (load testing) yang mencakup tiga aspek utama, yaitu throughput (Requests Per Second), latensi, dan penggunaan sumber daya, terdapat perbedaan performa yang konsisten antara Go Gin dan Python Flask dalam menangani 500 pengguna secara bersamaan pada aplikasi Alumni Circle. Dari sisi throughput, Go Gin memproses total 55.739 permintaan dengan tingkat keberhasilan 100% dan rata-rata 185,88 RPS, sementara Flask hanya mampu menyelesaikan 29.201 permintaan dengan 7.322 di antaranya gagal dan rata-rata RPS hanya 97,36. Perbedaan ini semakin terlihat pada endpoint dengan beban data besar seperti /alumni dan /event, di mana Flask mengalami penurunan performa yang drastis hingga di bawah 1 RPS, sedangkan Go Gin tetap stabil di kisaran 36–37 RPS. Hal ini menunjukkan bahwa model concurrency yang dimiliki Go secara arsitektural memang dirancang untuk menangani banyak koneksi secara bersamaan tanpa degradasi yang berarti. Dari sisi latensi, Go Gin mencatat rata-rata waktu respons 468,18 ms, hampir lima kali lebih cepat dibanding Flask yang menyentuh 2.332,72 ms. Selisih sebesar lebih dari 1,8 detik pada kondisi rata-rata ini sudah cukup untuk mengganggu pengalaman pengguna secara nyata, terlebih pada aplikasi mobile yang menuntut respons cepat. Pada kondisi ekstrem, Flask bahkan mencatatkan waktu tunggu maksimal hingga 132.860 ms yang mengindikasikan

DOI: https://doi.org/10.31004/riggs.v5i1.7664

sistem hampir tidak responsif, berbanding jauh dengan Go Gin yang mampu meredam lonjakan di angka 43.117 ms tanpa menghasilkan satu pun error. Kondisi ini memperlihatkan bahwa Flask tidak hanya lebih lambat secara rata-rata, tetapi juga jauh lebih rentan terhadap lonjakan beban yang tidak terduga. Dari sisi penggunaan sumber daya, kedua framework menunjukkan rata-rata CPU yang relatif sebanding, namun Go Gin lebih stabil di beban puncak dengan CPU tertinggi hanya 41,21% dibanding Flask yang mencapai 60,82%. Keunggulan Go Gin lebih nyata pada efisiensi memori, dengan rata-rata pemakaian 53,54 MB berbanding 114,65 MB milik Flask, atau sekitar 2,1 kali lebih hemat. Efisiensi memori ini bukan hal yang sepele dalam konteks skalabilitas, karena konsumsi memori yang lebih rendah berarti satu unit server dapat menjalankan lebih banyak instance aplikasi secara bersamaan, yang pada akhirnya berdampak langsung pada penghematan biaya infrastruktur jangka panjang. Secara keseluruhan, Go Gin terbukti lebih unggul dalam ketiga aspek penilaian pengujian. Kombinasi antara throughput yang tinggi, latensi yang rendah dan stabil, serta konsumsi memori yang efisien menjadikan Go Gin sebagai pilihan arsitektur backend yang lebih andal dan skalabel untuk aplikasi Alumni Circle, terutama dalam menghadapi pertumbuhan jumlah pengguna di masa mendatang. Meski demikian, perlu diakui bahwa Python Flask tetap memiliki nilai lebih dari sisi kemudahan pengembangan dan ekosistem library yang luas, sehingga pemilihan framework pada akhirnya harus disesuaikan dengan prioritas tim pengembang antara kecepatan pengembangan dan kebutuhan performa produksi.

Referensi

[1] Y. N. Sari and C. Mukmin, "Pengembangan Sistem Informasi Tracer Study Pada SMK Muhammadiyah 1 Palembang," *Journal of Information Technology Ampera*, vol. 3, no. 2, pp. 94–107, 2022.

[2] S. Sadikin, D. Sujana, and E. D. Ariyani, "Studi Penelusuran (Tracer Studi) Alumni sebagai Sarana Pemantauan Serapan Lulusan Di Politeknik Manufaktur Bandung," *Jurnal Ilmiah Manajemen, Ekonomi, & Akuntansi (MEA)*, 2023.

[3] R. A. Asri, R. Albar, M. B. Wibawa, and M. A. Ilya, "Analisis Pengaruh Sistem Informasi Manajemen Alumni Berbasis Web Terhadap Peningkatan Keterlibatan Alumni Di MAN N 1 Tapaktuan," *JOURNAL OF INFORMATICS AND COMPUTER SCIENCE*, vol. 11, no. 2, pp. 30–38, 2025.

[4] G. R. M. L. B. Aritonang, M. A. Hasan, M. K. Ridwan, M. N. Iman, R. C. P. Silalahi, and R. S. Sipayung, "OPTIMASI QUERY SQL SERVER DENGAN TEKNIK INDEXING DAN PERFORMANCE MONITORING," *JATI (Jurnal Mahasiswa Teknik Informatika)*, vol. 9, no. 2, pp. 3094–3099, 2025.

[5] I. Sahputra, "Analisis Performa Website Perpustakaan Universitas Malikussaleh Menggunakan Metode Load Testing dengan Apache JMeter," *JATISI*, vol. 12, no. 4, 2025.

[6] A. Iskandar, M. I. Sarif, M. F. Hafiz, D. R. Harahap, and S. S. T. Purba, "Analisis Komparatif Kinerja Laravel Octane dan Webman pada Layanan API Berbasis Worker Model PHP," *Jurnal Komputer Teknologi Informasi Sistem Informasi (JUKTISI)*, vol. 4, no. 3, pp. 1583–1590, 2026.

[7] B. T. Hanggara, M. H. Nasrullah, and D. Pramono, "Analisis Perbandingan Performa Framework NestJS dan Lumen Pada Studi Kasus Aplikasi Berbasis REST API," *J-INTECH*, vol. 12, no. 1, pp. 181–189, 2024.

[8] O. ZHULKOVSKYI, I. ZHULKOVSKA, H. VOKHMIANIN, and A. TKACH, "COMPARATIVE ANALYSIS OF COMPUTATIONAL PERFORMANCE OF MODERN PROGRAMMING LANGUAGES," *Computer systems and information technologies*, no. 2, pp. 104–111, 2025.

[9] A. S. Azzahidi, B. Wijayanto, and A. Darmawan, "Performance Evaluation of Backend Frameworks for REST API: A Comparative Study of Spring Boot, Flask, Express. js, Laravel FrankenPHP, and Gin," *Jurnal Teknik Informatika (Jutif)*, vol. 6, no. 4, pp. 2405–2419, 2025.

[10] R. Kouatly, O. Demirci, and D. Das, "Comparison of Object-Relational Mapping Frameworks on C#, Python, Go, and Node.js using MSSQL With A Case Study," in *2025 International Conference Automatics and Informatics (ICAI)*, 2025, pp. 430–437. doi: 10.1109/ICAI67591.2025.11324528.

[11] F. Effendy, Taufik, and B. Adhilaksono, "Performance comparison of web backend and database: A case study of node. js, Golang and MySQL, Mongo DB," *Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science)*, vol. 14, no. 6, pp. 1955–1961, 2021.

[12] L. Fernando and M. M. Engel, "Comparative Performance Benchmarking of WebSocket Libraries on Node. js and Golang," *Sinkron: jurnal dan penelitian teknik informatika*, vol. 9, no. 4, 2025.

[13] H. Anderstedt and M. Wifvesson, "Benchmarking and Load Testing a Dynamic CRM Architecture," *LU-CS/HBG-EX*, 2025.

[14] C. López, D. Morato, E. Magaña, and M. Izal, "Validation of HTTP response time from network traffic as an alternative to web browser instrumentation," *IEEE Transactions on Network and Service Management*, vol. 19, no. 2, pp. 976–990, 2021.

[15] R. Malhotra, A. Bansal, and M. Kessentini, "Deployment and performance monitoring of docker based federated learning framework for software defect prediction," *Cluster Comput.*, vol. 27, no. 5, pp. 6039–6057, 2024.

DOI: https://doi.org/10.31004/riggs.v5i1.7664