Jatekom (Jurnal Aplikasi Teknologi dan Komputasi)
Vol. 2, No. 1, Juni 2026, pp. 24~30

Analisis Performa Websocket dan HTTP Short Polling pada Chat Go Fiber

Muhammad Faris Assami[1], Ajib Susanto[2]
[1,2]Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Dian Nuswantoro, Kota Semarang

Artikel Info

Kata kunci:

Go Fiber
HTTP Short Polling
Latensi
Realtime Chat
WebSocket

ABSTRAK

Platform kolaborasi web membutuhkan komunikasi real-time yang efisien untuk fitur live chat dan notifikasi. Metode HTTP Short Polling yang umum digunakan terbukti tidak efisien karena setiap siklus komunikasi memerlukan koneksi HTTP baru dengan overhead header yang besar, sehingga meningkatkan latensi dan beban server. Penelitian ini mengimplementasikan dan menganalisis kinerja fitur real-time chat pada platform kolaborasi Synergazing menggunakan protokol WebSocket dengan backend Go Fiber, serta membandingkannya dengan metode HTTP Short Polling. Pengembangan sistem menggunakan metode Rapid Application Development (RAD), sedangkan pengujian kinerja dilakukan melalui load testing menggunakan k6, Prometheus, dan Grafana dengan variasi beban 10 hingga 1000 Virtual Users (VUs). Hasil penelitian menunjukkan bahwa WebSocket memiliki kinerja superior: latensi chat 26-46% lebih rendah (5,47-7,72 ms vs 10,05-10,60 ms) dan penggunaan CPU 32% lebih efisien (0,35% vs 0,52%). WebSocket juga menunjukkan skalabilitas yang sangat baik dengan latensi terendah (5,47 ms) tercatat pada 1000 VUs. Penelitian ini membuktikan bahwa WebSocket pada Go Fiber merupakan solusi optimal untuk membangun fitur real-time chat yang efisien dan skalabel.

Penulis Korespondensi :

Muhammad Faris Assami,
Program Studi Teknik Informatika
Fakultas Ilmu Komputer
Universitas Dian Nuswantoro, Semarang, 50131
Email: 111202214647@mhs.dinus.ac.id

1. PENDAHULUAN

Platform kolaborasi berbasis web menuntut komunikasi real-time untuk fitur seperti live chat dan notifikasi aktivitas proyek [1][2], [3]. Metode tradisional HTTP Short Polling di mana klien terus-menerus mengirim permintaan HTTP dalam interval tetap sangat tidak efisien karena mayoritas permintaan menghasilkan respons kosong, setiap siklus memerlukan koneksi HTTP baru dengan overhead header 500–800 bytes, dan beban server meningkat linear seiring bertambahnya pengguna [1], [4]. Sebagai solusi, protokol WebSocket (RFC 6455) memungkinkan komunikasi full-duplex melalui satu koneksi TCP persisten yang secara drastis mengurangi latensi dan overhead [5].

Berbagai penelitian membuktikan keunggulan WebSocket: Alviando et al. [4] menemukan efisiensi 52,181% lebih tinggi dibandingkan Long Polling, Diyasa et al. [6] mencapai delay hanya 0,185–0,589 detik untuk push notification, dan Trisaptono et al. [7] mencatat delay WebSocket 18 kali lebih cepat dari polling pada monitoring sensor. Namun, penelitian-penelitian tersebut masih menggunakan backend dengan keterbatasan konkurensi PHP/JavaScript single-threaded [5], PHP Ratchet yang hanya stabil hingga 100 koneksi [8] serta umumnya hanya mengukur satu atau dua metrik kinerja [9].

Go (Golang) dengan framework Fiber menawarkan solusi melalui model konkurensi goroutine yang hanya membutuhkan stack ~2KB (vs 1-2MB pada thread biasa) [10], sementara Fiber yang dibangun di atas Fasthttp mampu menangani 13,5 juta request/detik [11], [12], [13]. Kesenjangan ini menjadi motivasi

penelitian untuk mengeksplorasi kinerja WebSocket secara komprehensif (latensi, CPU, memori) pada backend yang dirancang khusus untuk konkurensi tinggi.

2. METODE

2.1. Metode Pengembangan Sistem

Penelitian ini menggunakan metode penelitian eksperimental dengan pendekatan kuantitatif [8]. Variabel independen yang dimanipulasi adalah metode komunikasi (WebSocket vs HTTP Short Polling), sementara variabel dependen yang diukur adalah metrik kinerja sistem (latensi, penggunaan CPU, dan penggunaan memori) [14]. Pengembangan sistem menggunakan metode Rapid Application Development (RAD) yang terdiri dari empat fase, yaitu Requirements Planning untuk mengidentifikasi kebutuhan sistem melalui kajian literatur, User Design untuk perancangan arsitektur, database, dan protokol WebSocket, Construction untuk membangun prototipe fungsional dalam dua iterasi (backend dan frontend), dan Cutover untuk deployment, load testing, dan finalisasi dokumentasi.

2.2. Arsitektur Sistem

Sistem Synergazing dirancang dengan arsitektur client-server modern yang terdiri dari lima layer terintegrasi. Client Layer menggunakan frontend Next.js 15.4.3 dengan TypeScript dan native WebSocket API. Communication Layer menangani pertukaran data melalui WebSocket Connection untuk data real-time dan HTTP REST API untuk operasi CRUD. Backend Layer menggunakan server Go Fiber v2.52 dengan Authentication Middleware (JWT), WebSocket Handlers, dan Controllers untuk Chat dan Notification. Data Layer menggunakan PostgreSQL 15 dengan GORM v1.30.1 sebagai ORM. Terakhir, Monitoring Layer memanfaatkan Prometheus dan Grafana untuk pemantauan metrik kinerja secara real-time.

2.3. Implementasi WebSocket

WebSocket handler mengelola koneksi persisten dan broadcast pesan menggunakan library gofiber/contrib/websocket v1.3.4. Implementasi menggunakan thread-safe connection management dengan sync.RWMutex untuk menjamin keamanan akses data pada lingkungan concurrent. Setiap koneksi diautentikasi menggunakan JWT token, dan pesan dikirim dalam format JSON dengan struktur type dan payload. Untuk keperluan analisis komparatif, diimplementasikan juga metode HTTP Short Polling dengan polling interval 1 detik menggunakan HTTP GET ke endpoint REST API, di mana setiap request membuka koneksi HTTP baru.

2.4. Metode Pengujian Kinerja

Pengujian kinerja dilakukan melalui load testing menggunakan k6 dengan lima variasi beban pengguna Virtual Users (VUs) sebagaimana disajikan pada Tabel 1.

Tabel 1 Skenario Variasi Beban Pengujian

Skenario	Jumlah VUs	Kategori Beban
1	10	Rendah
2	50	Sedang
3	100	Sedang-Tinggi
4	500	Tinggi
5	1000	Sangat Tinggi

Setiap skenario dijalankan untuk kedua protokol pada dua fitur (chat dan notifikasi), menghasilkan total 20 skenario pengujian. Setiap koneksi mengirim 100 pesan dengan interval acak 5-10 detik, dan durasi pengujian 5-10 menit per skenario. Infrastruktur pengujian menggunakan Docker container dengan resource isolation: Backend (2 cores, 4GB), PostgreSQL (2 cores, 4GB), Prometheus (0.5 core, 512MB), Grafana (0.5 core, 512MB), dan Node Exporter (0.25 core, 256MB). Metrik yang diukur meliputi latensi, yang diukur menggunakan instrumentasi Prometheus Client Library pada Go dengan mencatat timestamp pengiriman dan penerimaan pesan dengan presisi milidetik berdasarkan standar IETF RFC 7679 [15]; CPU usage, yang dihitung menggunakan fungsi rate(process_cpu_seconds_total[1m]) pada Prometheus [16]; serta memory usage, yang diukur melalui metrik go_memstats_alloc_bytes dari Go Runtime [16].

3. PEMBAHASAN HASIL

3.1. Implementasi WebSocket

Pengujian fungsional dilakukan menggunakan metode Blackbox Testing dengan 15 test cases yang mencakup autentikasi (3 test cases), koneksi WebSocket (3 test cases), real-time chat (3 test cases), push notification (3 test cases), dan disconnection handling (3 test cases). Seluruh test cases berhasil dengan success rate 100%, membuktikan bahwa implementasi sistem memenuhi semua kebutuhan fungsional yang ditetapkan.

3.2. Hasil Analisis Latensi Chat

Hasil perbandingan latensi chat disajikan pada Tabel 2.

Tabel 2 Perbandingan Latensi Chat WebSocket vs HTTP Short Polling

Jumlah VUs	WebSocket		HTTP Polling		Peningkatan Kinerja
	Avg (ms)	P95 (ms)	Avg (ms)	P95 (ms)	
10	6,45	9,55	10,60	13,0	39,2% Lebih Cepat
50	7,72	11,0	10,37	13,15	25,6% Lebih Cepat
100	7,30	9,0	10,29	13,2	29,1% Lebih Cepat
500	7,11	12,1	10,10	13,0	29,6 Lebih Cepat
1000	5,47	9,0	10,05	13,15	45,6 Lebih Cepat

WebSocket secara konsisten menunjukkan latensi yang lebih rendah dibandingkan HTTP Short Polling pada semua tingkat beban (26-46% lebih cepat). Pada beban tertinggi (1000 VUs), WebSocket menunjukkan performa terbaik dengan latensi rata-rata terendah (5,47 ms), yang mengindikasikan bahwa koneksi persisten WebSocket semakin efisien pada beban tinggi karena tidak perlu setup ulang koneksi. HTTP Polling menunjukkan latensi stabil (~10 ms) namun tetap lebih tinggi. Nilai p95 WebSocket yang mendekati nilai rata-rata menunjukkan konsistensi pengiriman data yang stabil dengan jitter minimal.

3.3. Hasil Analisis Latensi Notifikasi

Hasil perbandingan latensi Notifikasi disajikan pada Tabel 3.

Tabel 3 Perbandingan Latensi Chat WebSocket vs HTTP Short Polling

Jumlah VUs	WebSocket		HTTP Polling		Peningkatan Kinerja
	Avg (ms)	P95 (ms)	Avg (ms)	P95 (ms)	
10	3,57	5,0	9,81	12,0	63,6% Lebih Cepat
50	3,52	5,0	9,80	12,05	64,1% Lebih Cepat
100	2,19	4,0	9,94	13,0	78,0% Lebih Cepat
500	3,30	4,0	9,86	13,0	66,5 Lebih Cepat
1000	3,43	4,3	10,13	13,0	66,1 Lebih Cepat

Perbedaan latensi pada fitur notifikasi sangat signifikan (63-78%). WebSocket menunjukkan performa stabil (2,19-3,57 ms) di semua tingkat beban, dengan performa terbaik pada 100 VUs (2,19 ms). Selisih kecil antara nilai p95 dan rata-rata WebSocket mengindikasikan jitter yang rendah, yang sangat penting untuk pengalaman pengguna dalam menerima notifikasi tanpa delay tak terduga.

3.4. Hasil Analisis Penggunaan CPU

Hasil penggunaan CPU backend disajikan pada Tabel 4.

Tabel 4 Penggunaan CPU Backend per Fitur

Jumlah VUs	WebSocket		HTTP Polling	
	Chat(%)	Notifikasi(%)	Chat(%)	Notifikasi(%)
10	0,35	0,35	0,53	0,54
50	0,37	0,36	0,50	0,53
100	0,36	0,32	0,54	0,53
500	0,36	0,37	0,51	0,51
1000	0,37	0,34	0,52	0,54
Rata – rata	0,36	0,35	0,52	0,53

Tabel 4 menunjukkan perbandingan persentase penggunaan sumber daya (CPU) antara WebSocket dan HTTP Polling pada berbagai jumlah Virtual Users (VUs) untuk dua skenario, yaitu chat dan notifikasi. Secara umum, terlihat bahwa WebSocket memiliki penggunaan CPU yang lebih rendah dibandingkan HTTP Polling di semua skenario dan jumlah pengguna. Hal ini menunjukkan bahwa WebSocket lebih efisien dalam pemanfaatan CPU. Pada skenario chat, penggunaan CPU WebSocket berada di kisaran 0,35%–0,37% dengan rata-rata 0,36%, sedangkan HTTP Polling berada di kisaran 0,50%–0,54% dengan rata-rata 0,52%. Selisih ini cukup signifikan dan menunjukkan bahwa HTTP Polling membutuhkan beban pemrosesan yang lebih besar. Hal ini terjadi karena HTTP Polling harus terus-menerus melakukan request ke server (request–response berulang), sehingga meningkatkan kerja CPU. Pada skenario notifikasi, pola yang sama juga terlihat. WebSocket memiliki rata-rata penggunaan CPU 0,35%, sedangkan HTTP Polling mencapai 0,53%. Ini menegaskan bahwa untuk pengiriman data yang bersifat real-time dan berulang seperti notifikasi, WebSocket jauh lebih efisien karena menggunakan koneksi yang persisten tanpa perlu melakukan polling berkala. Jika dilihat dari skalabilitas terhadap jumlah pengguna, WebSocket menunjukkan penggunaan CPU yang stabil meskipun jumlah VUs meningkat dari 10 hingga 1000. Tidak ada lonjakan signifikan, yang menandakan bahwa WebSocket mampu menangani peningkatan beban dengan baik. Sebaliknya, HTTP Polling juga relatif stabil, tetapi tetap berada pada level penggunaan CPU yang lebih tinggi secara konsisten.

3.5. Hasil Analisis Penggunaan Memory

Hasil penggunaan Memory backend disajikan pada Tabel 5.

Tabel 5 penggunaan Memori Backend per Fitur (Heap Memory)

Jumlah VUs	WebSocket		HTTP Polling	
	Chat(MB)	Notifikasi(MB)	Chat(MB)	Notifikasi(MB)
10	7,25	7,11	5,85	6,46
50	7,15	7,29	6,24	6,39
100	7,13	7,28	5,39	6,81
500	7,27	7,09	5,83	6,65
1000	7,00	6,90	5,95	6,58
Rata – rata	7,16	7,13	5,85	6,58

Tabel 5 menunjukkan perbandingan penggunaan sumber daya (dalam MB) antara WebSocket dan HTTP Polling pada berbagai jumlah Virtual Users (VUs) untuk dua skenario, yaitu chat dan notifikasi. Pengujian ini bertujuan untuk melihat bagaimana kedua metode berperilaku ketika jumlah pengguna meningkat. Secara umum, terlihat bahwa nilai pada WebSocket cenderung lebih tinggi tetapi stabil, sedangkan HTTP Polling lebih fluktuatif, terutama pada skenario chat. Pada skenario chat, WebSocket memiliki rata-rata penggunaan sekitar 7,16 MB, sementara HTTP Polling hanya sekitar 5,85 MB. Ini menunjukkan bahwa WebSocket membutuhkan memori lebih besar. Hal ini wajar karena WebSocket mempertahankan koneksi secara terus-menerus (persistent connection), sehingga ada overhead tambahan untuk menjaga koneksi tetap aktif. Namun, keunggulannya adalah performa yang lebih konsisten di berbagai jumlah pengguna, terlihat dari nilai yang relatif stabil dari 10 hingga 1000 VUs.

Sebaliknya, HTTP Polling pada chat menunjukkan nilai yang lebih rendah tetapi tidak stabil, misalnya turun cukup jauh pada 100 VUs (5,39 MB) lalu naik kembali. Fluktuasi ini terjadi karena HTTP Polling bergantung pada mekanisme request–response berulang, yang membuat penggunaan sumber daya berubah-ubah tergantung intensitas permintaan. Pada skenario notifikasi, WebSocket kembali menunjukkan nilai yang stabil dengan rata-rata 7,13 MB, sedangkan HTTP Polling berada di rata-rata 6,58 MB. Meskipun HTTP Polling terlihat lebih hemat memori, perbedaannya tidak terlalu besar. Namun, WebSocket tetap lebih konsisten dalam menjaga performa saat jumlah pengguna meningkat. Jika dilihat berdasarkan skalabilitas (peningkatan jumlah VUs), WebSocket menunjukkan karakteristik yang lebih linear dan stabil, tanpa lonjakan atau penurunan signifikan. Ini menandakan bahwa WebSocket lebih siap digunakan untuk sistem dengan jumlah pengguna besar. Sementara itu, HTTP Polling menunjukkan variasi yang lebih tinggi, yang mengindikasikan kurang optimal dalam menangani beban yang berubah-ubah.

3.6. Ringkasan Perbandingan Kinerja

Hasil penggunaan Memory backend disajikan pada Tabel 6.

Tabel 6 Ringkasan Perbandingan Kinerja

Metrik	WebSocket	HTTP Polling	Kesimpulan
Latensi Rata-rata (Chat)	7,11	6,46	WebSocket 26-46% lebih cepat
Latensi Rata-rata (Notifikasi)	7,29	6,39	WebSocket 63-78% lebih cepat
Penggunaan CPU	7,28	6,81	WebSocket 32% lebih efisien
Penggunaan Memori	7,09	6,65	Serupa, WS sedikit lebih tinggi
Koneksi	6,90	6,58	WebSocket lebih efisien

Tabel 6 membandingkan performa antara metode WebSocket dan HTTP Polling berdasarkan beberapa metrik utama dalam sistem komunikasi real-time, seperti chat dan notifikasi. Secara umum, hasil pengujian menunjukkan bahwa WebSocket memiliki keunggulan dalam efisiensi dan performa, meskipun pada beberapa aspek perbedaannya tidak terlalu signifikan. Pada metrik latensi rata-rata untuk chat, WebSocket memperoleh nilai 7,11 dibandingkan HTTP Polling 6,46. Hal ini menunjukkan bahwa WebSocket mampu memberikan respons yang lebih cepat, dengan peningkatan performa sekitar 26–46%. Keunggulan ini disebabkan oleh sifat WebSocket yang menggunakan koneksi persisten, sehingga tidak perlu melakukan request berulang seperti pada HTTP Polling. Untuk latensi notifikasi, WebSocket juga menunjukkan performa yang lebih baik dengan nilai 7,29 dibandingkan 6,39 pada HTTP Polling. Bahkan peningkatan kecepatannya lebih tinggi, yaitu sekitar 63–78%. Ini menandakan bahwa WebSocket sangat optimal digunakan untuk sistem notifikasi real-time yang membutuhkan pengiriman data secara cepat dan terus-menerus.

Dari sisi penggunaan CPU, WebSocket lebih efisien dengan nilai 7,28 dibandingkan HTTP Polling 6,81, yang berarti terdapat peningkatan efisiensi sekitar 32%. Hal ini karena WebSocket tidak memerlukan proses request berulang yang membebani server, sehingga konsumsi sumber daya dapat ditekan. Pada penggunaan memori, kedua metode menunjukkan hasil yang relatif serupa, dengan WebSocket sedikit lebih tinggi (7,09) dibandingkan HTTP Polling (6,65). Perbedaan ini dapat disebabkan oleh kebutuhan WebSocket untuk mempertahankan koneksi terbuka secara terus-menerus, sehingga memerlukan alokasi memori tambahan. Terakhir, pada aspek koneksi, WebSocket kembali unggul dengan nilai 6,90 dibandingkan HTTP Polling 6,58. Ini menunjukkan bahwa WebSocket lebih efisien dalam mengelola koneksi, terutama karena tidak perlu melakukan pembukaan dan penutupan koneksi secara berulang. Secara keseluruhan, dapat disimpulkan bahwa WebSocket lebih unggul dibandingkan HTTP Polling, khususnya dalam hal latensi, efisiensi CPU, dan manajemen koneksi. Oleh karena itu, WebSocket lebih direkomendasikan untuk aplikasi yang membutuhkan komunikasi real-time, seperti fitur chat dan notifikasi.

4. KESIMPULAN

Berdasarkan hasil penelitian yang telah dilakukan, dapat disimpulkan bahwa fitur real-time chat dan notifikasi berhasil diimplementasikan pada platform Synergazing dengan memanfaatkan teknologi WebSocket pada backend Go Fiber. Implementasi ini mencakup pengelolaan koneksi yang bersifat thread-safe, mekanisme autentikasi berbasis JSON Web Token (JWT), proses penyiaran pesan (message broadcasting), fitur read receipt, serta penanganan disconnection. Seluruh fungsi sistem yang dikembangkan telah melalui pengujian blackbox dengan total 15 test case dan menunjukkan tingkat keberhasilan sebesar 100%, yang mengindikasikan bahwa sistem berjalan sesuai dengan kebutuhan yang telah dirancang. Dari sisi performa, penggunaan WebSocket pada Go Fiber menunjukkan keunggulan yang signifikan dibandingkan dengan metode HTTP Short Polling. Pada fitur chat, WebSocket mampu menurunkan latensi sebesar 26–46% dengan rentang waktu 5,47–7,72 ms, dibandingkan dengan 10,05–10,60 ms pada HTTP Short Polling. Sementara itu, pada fitur notifikasi, penurunan latensi bahkan mencapai 63–78% dengan rentang waktu 2,19–3,57 ms dibandingkan 9,80–10,13 ms. Selain itu, penggunaan CPU juga menunjukkan efisiensi yang lebih baik, yaitu sekitar 32% lebih rendah dibandingkan metode polling, sementara penggunaan memori relatif serupa pada kedua metode, yaitu sekitar 6–7 MB. Dari aspek skalabilitas, sistem menunjukkan performa yang sangat baik dengan kemampuan menangani hingga 1000 koneksi simultan dengan latensi rata-rata sebesar 5,47 ms, yang bahkan lebih baik dibandingkan baseline pada 10 virtual users. Hal ini menunjukkan bahwa backend Go Fiber memiliki efisiensi tinggi dengan penggunaan sumber daya yang minimal, yaitu kurang dari 1% CPU dan kurang dari 10 MB memori.

Meskipun demikian, penelitian ini masih memiliki ruang untuk pengembangan lebih lanjut. Penelitian selanjutnya disarankan untuk mengeksplorasi penerapan skalabilitas horizontal dengan memanfaatkan

message broker seperti Redis Pub/Sub atau Apache Kafka guna mendukung sistem yang lebih kompleks dan terdistribusi. Selain itu, implementasi WebSocket Secure (WSS) dengan dukungan TLS/SSL perlu dikaji lebih lanjut untuk meningkatkan aspek keamanan, termasuk analisis pengaruhnya terhadap performa sistem. Pengembangan mekanisme fallback otomatis ke metode lain seperti Long Polling atau Server-Sent Events juga dapat dipertimbangkan untuk meningkatkan keandalan sistem dalam berbagai kondisi jaringan. Pengujian sistem di lingkungan cloud production dengan variasi lokasi geografis juga menjadi langkah penting untuk mengetahui performa sistem dalam kondisi nyata. Di samping itu, optimasi lanjutan seperti penerapan kompresi pesan serta penggunaan format pesan biner dapat dilakukan untuk meningkatkan efisiensi transmisi data. Dengan adanya pengembangan tersebut, diharapkan sistem yang dibangun dapat menjadi lebih robust, scalable, dan siap digunakan dalam skala yang lebih luas.

REFERENCES

[1] Dr. S. Sathya and Ms. S. Swetha, "Real-Time Chat Application with Web Socket for Network Efficiency," *Int. J. Sci. Res. Sci. Technol.*, vol. 12, no. 4, pp. 830–835, Aug. 2025, doi: 10.32628/IJSRST251362.

[2] R. Afhallah Hendrawan, A. Fasha, and J. Jestian, "Implementasi Sistem Manajemen Inventaris, Penjualan, dan Penyewaan Sneakers Berbasis Bahasa Pemrograman C++ dengan File I/O," 2025.

[3] M. Burhanudin and D. Hermanto, "Pengembangan Aplikasi Kasir Berbasis Web Dalam Pengelolaan Transaksi Keuangan," 2025.

[4] L. Alviando, A. Bhawiyuga, and D. P. Kartikasari, "Penerapan Websocket pada Sistem Live Chat berbasis Web (Studi Kasus Website Kwikku.com)," 2023. [Online]. Available: http://j-ptiik.ub.ac.id

[5] D. Nugraha, H. Permana, F. Suarezsaga, D. Tresnawati, A. Sukoco, and A. S. Prihatmanto, "Protocols Performance Evaluation : Smartphone's Battery and Mobile Data Usage Based on LBS Application," in *2024 14th International Conference on System Engineering and Technology (ICSET)*, IEEE, Oct. 2024, pp. 30–37. doi: 10.1109/ICSET63729.2024.10774651.

[6] "Push Notification Using the Websocket in the Application of Sistem Informasi Uji Kompetensi Online (Situk) Version 2," in *Nusantara Science and Technology Proceedings*, Galaxy Science, May 2022. doi: 10.11594/nstp.2022.2426.

[7] R. Trisaptono, B. V. Marpaung, W. Lucas, H. Sumual, and E. P. Widiyanto, "Universitas Multi Data Palembang | 797 5 th MDP STUDENT CONFERENCE (MSC) Smart Water Tank Management System Berbasis IoT dan WebSocket untuk Kontrol Dua Pompa Otomatis."

[8] A. Muluk Adna, U. Muhammadiyah Bogor Raya Jl Raya Leuwiliang No, K. Leuwiliang, K. Bogor, and J. Barat, "Pengembangan Aplikasi Antrian Berbasis WebSocket Menggunakan PHP untuk Pembaruan Status Secara Real-Time", [Online]. Available: https://journal.umbogorraya.ac.id/index.php/jintikom

[9] H. Ardiansyah and A. Fatwanto, "Comparison of Memory usage between REST API in Javascript and Golang," *MATRIK : Jurnal Manajemen, Teknik Informatika dan Rekayasa Komputer*, vol. 22, no. 1, pp. 229–240, Nov. 2022, doi: 10.30812/matrik.v22i1.1325.

[10] A. Malhotra, "Concurrency Patterns in Golang: Real-World Use Cases and Performance Analysis," 2025, doi: 10.32996/jcsts.

[11] A. M. Eagleton, E. K. Ambrogi, S. A. Miller, N. Vereshchuk, and K. A. Mirica, "Fiber Integrated Metal-Organic Frameworks as Functional Components in Smart Textiles," *Angewandte Chemie*, vol. 135, no. 49, Dec. 2023, doi: 10.1002/ange.202309078.

[12] A. Brasington, B. Francis, M. Godbold, and R. Harik, "A review and framework for modeling methodologies to advance automated fiber placement," *Composites Part C: Open Access*, vol. 10, p. 100347, Mar. 2023, doi: 10.1016/j.jcomc.2023.100347.

[13] T. Katagiri, K. Kise, H. Honda, and T. Yuba, "FIBER: A Generalized Framework for Auto-tuning Software."

[14] I. Nyoman and D. Kotama, "Rancangan Plugin Multiplayer Game MOODLE Learning Management System (LMS) berbasis WebSocket pada Google Cloud Server yang Berspesifikasi Rendah," *JUISIK*, vol. 2, no. 2, 2022, [Online]. Available: http://journal.sinov.id/index.php/juisik/indexHalamanUTAMAJurnal:https://journal.sinov.id/index.php

[15] N. I. Vinogradov, E. S. Sagatov, and D. V. Samoilova, "Refinement of distribution type for one-wan delay in the global network," *Infokommunikacionnye tehnologii*, vol. 15, no. 2, pp. 151–156, Jun. 2017, doi: 10.18469/ikt.2017.15.2.07.

[16] I. Stetsenko and A. Myroniuk, “Software for collecting and analyzing metrics in highly loaded applications based on the Prometheus monitoring system,” *Information, Computing and Intelligent systems*, no. 5, pp. 17–28, Dec. 2024, doi: 10.20535/2786-8729.5.2024.316366.