Pengembangan Sistem Manajemen Gudang Berbasis Web Dengan *Event-Driven Architecture*

Candra Bagus Ainur Rochman[1], I Made Suartana [2]

[1,2] Teknik Informatika, Universitas Negeri Surabaya
[1]candrabagus.22007@mhs.unesa.ac.id
[2]madesuartana@unesa.ac.id

Abstrak*— Sistem manajemen gudang berbasis web rentan mengalami degradasi kinerja saat menangani lonjakan transaksi secara bersamaan, terutama ketika arsitektur yang digunakan memproses seluruh alur bisnis secara sinkron dalam satu jalur permintaan. Penelitian ini bertujuan mengimplementasikan Event-Driven Architecture (EDA) sebagai penyesuaian arsitektur pada sistem WMS berbasis web dan mengevaluasi pengaruhnya terhadap latensi serta *throughput* dibandingkan arsitektur modular monolith. Penelitian menggunakan desain eksperimen kuantitatif before after pada sistem WMS yang dibangun dengan stack Laravel React MySQL dan dijalankan di lingkungan Docker Kubernetes. Penerapan EDA dilakukan dengan memindahkan proses pendukung (pencatatan log, notifikasi, dan pembaruan *dashboard*) ke pemrosesan asinkron menggunakan Laravel Queue dengan Redis sebagai *message broker*. Pengujian dilakukan pada tiga use case, yaitu *Inbound*/Receiving, Stock Adjustment, dan *Outbound* Confirmation, dengan beban bertahap 50, 100, dan 200 request per second menggunakan k6, serta pemantauan sumber daya melalui Hasil pengujian menunjukkan EDA unggul signifikan pada *Inbound* dan *Outbound*. Pada beban normal *Outbound*, latensi *end-to-end* turun dari 18,84 detik menjadi 204 milidetik dan *throughput* meningkat 55,09%. Pada *stress test* UC-03, *throughput* meningkat 645,95% dengan success rate 100%, sementara modular monolith mengalami kegagalan fungsional hingga 81% akibat database *lock contention*. Sebaliknya, EDA menunjukkan keterbatasan pada adjustment karena *overhead* antrian dan validasi duplikasi data memperparah kontesi basis data yang menggunakan *pessimistic locking*. Disimpulkan bahwa penerapan EDA secara inkremental efektif meningkatkan kinerja dan skalabilitas WMS, khususnya pada operasi dengan alur bisnis kompleks. Disarankan penerapan *eventual consistency* secara penuh dan penggantian *pessimistic locking* dengan *optimistic locking* untuk mengoptimalkan performa pada seluruh skenario.

Kata Kunci— **Warehouse Management System, Event-Driven Architecture, Modular Monolith, Latensi, *Throughput*, Skalabilitas.**

I. PENDAHULUAN

Warehouse Management System (WMS) berbasis web digunakan untuk mencatat pergerakan stok, mengendalikan alur penerimaan dan pengeluaran barang, serta menyediakan informasi inventori yang konsisten bagi pengambilan keputusan operasional [1]-[5]. Namun, ketika transaksi gudang terjadi secara bersamaan, WMS tidak hanya dituntut akurat, tetapi juga harus mempertahankan response time, throughput, dan reliability yang stabil. Sistem yang tidak dirancang untuk menghadapi pertumbuhan beban dapat mengalami peningkatan latensi, penurunan kapasitas pemrosesan, keterlambatan pembaruan data, atau error pada saat puncak aktivitas seperti distribusi, audit stok, dan peningkatan pesanan [6]-[8].

Modular monolith banyak digunakan sebagai arsitektur awal aplikasi web karena deployment, debugging, dan monitoring lebih sederhana, sementara pemisahan modul internal tetap membantu maintainability [9], [10]. Keterbatasannya muncul ketika proses utama dan proses pendukung masih dieksekusi secara sinkron dalam satu jalur request. Pada kondisi beban tinggi, pencatatan log, notifikasi, dan pembaruan dashboard dapat memperpanjang waktu tunggu pengguna serta menurunkan kapasitas aplikasi secara keseluruhan [11], [12].

Event-Driven Architecture (EDA) menawarkan pendekatan berbeda dengan memisahkan producer, broker atau queue, dan consumer sehingga komponen sistem dapat berinteraksi melalui event secara asynchronous [13], [14]. Dalam konteks WMS, pemrosesan kritis seperti validasi dan perubahan stok tetap dilakukan pada jalur utama, sedangkan proses pendukung dapat dipindahkan ke worker melalui message broker. Pola ini berpotensi memperpendek jalur transaksi, meningkatkan isolasi beban, dan memungkinkan scaling pada komponen worker tanpa harus menskalakan seluruh aplikasi.

Meskipun EDA banyak direkomendasikan untuk meningkatkan scalability dan resilience, penelitian terdahulu masih lebih banyak membahas konsep umum, sistem terdistribusi generik, atau domain selain WMS [15]-[18]. Studi evaluasi performa aplikasi web juga menunjukkan pentingnya pengukuran response time, throughput, dan success rate pada beban tinggi, tetapi belum secara spesifik menempatkan EDA sebagai transisi langsung dari modular monolith pada sistem WMS berbasis Laravel, React dan MySQL [19], [20]. Oleh karena itu, penelitian ini mengevaluasi penerapan EDA secara inkremental pada WMS berbasis web melalui pengujian before-after menggunakan metrik kuantitatif dan monitoring sumber daya.

Penelitian ini bertujuan mengimplementasikan EDA pada modul terpilih WMS berbasis web serta membandingkan kinerja sistem sebelum dan sesudah transisi arsitektur. Fokus evaluasi diarahkan pada latency p(95), throughput, success rate, dropped iterations, serta indikasi bottleneck dari pemantauan CPU dan memori.

II. METODE PENELITIAN

A. *Desain Penelitian*

Penelitian ini menggunakan desain eksperimen kuantitatif before-after pada satu sistem WMS yang sama. Kondisi before adalah arsitektur modular monolith, sedangkan kondisi after adalah EDA berbasis Laravel Queue dengan Redis sebagai message broker. Variabel bebas penelitian adalah model arsitektur sistem; variabel terikat meliputi latency p(95), throughput, success/error rate, dan dropped iterations; sedangkan variabel kontrol mencakup dataset awal, endpoint, tingkat beban RPS, resource container, dan lingkungan eksekusi Docker-Kubernetes. Desain ini digunakan agar

perubahan performa dapat dikaitkan dengan perubahan arsitektur, bukan perbedaan fungsi aplikasi atau data uji.

B. *Desain Arsitektur Sistem*

Sistem WMS mengelola tiga proses utama, yaitu inbound/receiving, stock adjustment, dan outbound confirmation. Pada logika inventori, aturan FEFO dan FIFO ditempatkan sebagai mekanisme pemilihan batch pada proses outbound: FEFO diprioritaskan untuk produk yang memiliki expired_date, sedangkan FIFO digunakan sebagai fallback ketika tanggal kedaluwarsa sama atau produk tidak memiliki atribut masa berlaku. Dengan demikian, implementasi FEFO/FIFO menjadi bagian dari logika domain WMS, sedangkan fokus utama penelitian tetap berada pada perubahan arsitektur pemrosesan.

Pada baseline modular monolith, validasi transaksi, pembaruan stok, pencatatan aktivitas, notifikasi, dan pembaruan data dashboard berjalan dalam satu alur sinkron. Pada arsitektur EDA, hanya proses kritis seperti validasi dan penyimpanan data yang tetap berada di jalur utama, sedangkan proses pendukung didelegasikan ke queue dan diproses oleh worker/listener secara asinkron. Ringkasan use case yang diuji ditunjukkan pada Tabel 1.

TABEL I
USE CASE DAN PERBEDAAN MEKANISME PENGUJIAN

Use case	Fokus pengujian	Mekanisme EDA
Use Case - 01 Inbound	Pencatatan barang masuk dan update stok	Log dan notifikasi diproses melalui queue
Use Case - 02 Adjustment	Perubahan stok intensif dan pembaruan rekap dashboard	Rekap dashboard diproses oleh listener latar belakang
Use Case - 03 Outbound	Pengeluaran barang dan konfirmasi transaksi	Notifikasi dan proses pendukung dilepas dari jalur respons pengguna

C. *Lingkungan dan Prosedur Pengujian*

Pengujian dilakukan menggunakan k6 dengan pola beban bertahap 50, 100, dan 200 request per second (RPS). Setiap use case dieksekusi pada dua kondisi arsitektur dengan dataset, endpoint, dan resource yang sama. Lingkungan pengujian menggunakan Docker dan Kubernetes dengan pembatasan resource agar perbandingan berlangsung pada kondisi yang konsisten. CPU dan memori dipantau melalui Grafana dan cAdvisor untuk membantu membaca indikasi bottleneck selama pengujian.

TABEL II
USE CASE DAN PERBEDAAN MEKANISME PENGUJIAN

Komponen	Konfigurasi
Stack aplikasi	Laravel backend, React frontend, MySQL database
Arsitektur pembanding	Modular monolith dan EDA berbasis Laravel Queue
Message broker	Redis
Lingkungan eksekusi	Docker dan Kubernetes
Resource limit	2 vCPU dan 2 GB RAM
Dataset awal	200 data barang dan 10 data pengguna
Load testing	k6 pada 50, 100, dan 200 RPS
Monitoring	Grafana dan cAdvisor

D. *Metrik Pengukuran*

Metrik utama yang digunakan adalah latency p(95), yaitu nilai latensi di mana 95% request memiliki waktu respons di bawah nilai tersebut; throughput, yaitu jumlah request sukses per detik; success atau error rate; serta dropped iterations. Pada skenario yang memiliki lebih dari satu tahap transaksi, pengukuran juga mencakup waktu end-to-end dari awal transaksi hingga proses konfirmasi selesai.

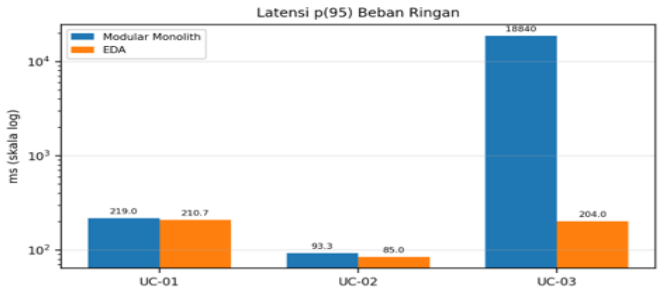
III. HASIL DAN PEMBAHASAN

A. *Hasil Pengujian Beban Ringan*

Pada beban ringan, penerapan EDA tidak selalu meningkatkan seluruh metrik secara merata, tetapi menunjukkan pola yang jelas sesuai karakteristik use case. Ringkasan perbandingan kuantitatif ditunjukkan pada Tabel 3, dan visualisasi latensi p(95) ditunjukkan pada Gambar 2.

TABEL III
RINGKASAN KOMPARATIF PADA BEBAN RINGAN

Use Case	Arsitektur	Throughput	Latensi (p95)	Tingkat Sukses	Kesimpulan
UC-01 ***(Inbound)***	Monolith	15.53 req/s	219.00 ms	99.53% *(16 error)*	EDA unggul tipis pada latensi dan reliabilitas.
	EDA	15.54 req/s	**210.69 ms**	**100.00 %**	
UC-02 ***(Adjustment)***	Monolith	7.77 req/s	93.34 ms	**97.14%**	EDA sedikit lebih cepat, tetapi menghasilkan *error rate* lebih tinggi.
	EDA	7.77 req/s	**85.00 ms**	95.52%	
UC-03 ***(Outbound)***	Monolith	10.02 req/s	18.84 s	535 *Dropped*	EDA unggul dominan; sukses memangkas latensi dari hitungan detik ke milidetik.
	EDA	**15.54 req/s**	**204.00 ms**	**0 *Dropped***	



Gbr. 2 Visualisasi latensi p(95) pada beban ringan. Sumbu y menggunakan skala logaritmik agar perbedaan UC-03 tetap terbaca.

Pada UC-01, throughput kedua arsitektur relatif setara, tetapi EDA menurunkan latensi end-to-end dari 219,00 ms menjadi 210,69 ms dan menghilangkan 16 error yang muncul pada modular monolith. Peningkatan ini bersifat moderat karena

proses inbound masih didominasi oleh validasi dan pembaruan stok yang tetap harus terjadi pada jalur utama.

Pada UC-02, EDA menurunkan latensi dari 93,34 ms menjadi 85,00 ms, tetapi success rate menurun dari 97,14% menjadi 95,52%. Hasil ini menunjukkan bahwa pemindahan proses dashboard ke background worker belum otomatis memperbaiki reliability ketika operasi utama masih menimbulkan kontesi basis data. Dengan kata lain, EDA membantu mempersingkat sebagian waktu komputasi, tetapi belum menyelesaikan akar bottleneck pada pembaruan stok intensif.

Pada UC-03, dampak EDA paling kuat karena proses pendukung pada konfirmasi outbound dapat dipisahkan secara efektif dari respons pengguna. Throughput meningkat dari 10,02 req/s menjadi 15,54 req/s, sedangkan latensi end-to-end turun dari 18,84 detik menjadi 204,00 ms. Penghapusan 535 dropped iterations pada modular monolith memperlihatkan bahwa pemisahan jalur notifikasi dan proses pendukung sangat relevan untuk operasi dengan alur bisnis panjang.

B. *Hasil Pengujian Stress Test*

Stress test dipisahkan per use case agar pembaca dapat melihat pola perubahan performa dari 50 hingga 200 RPS secara lebih jelas pada masing-masing skenario operasional.

1) Analisis UC-01 (Inbound/Receiving)

TABEL IV
HASIL STRESS TEST UC-01 (INBOUND/RECEIVING)

Beban	Arsitektur	Throughput	Latensi (E2E)	Error (%)	Request Dropped
50 RPS	Monolith	55,99 req/s	20,65 s	1,37%	1.434
	EDA	67,33 req/s	16,06 s	0,00%	782
100 RPS	Monolith	58,29 req/s	12,55 s	4,30%	6.664
	EDA	70,49 req/s	9,88 s	0,00%	5.930
200 RPS	Monolith	56,79 req/s	21,45 s	6,80%	22.852
	EDA	69,23 req/s	18,15 s	0,00%	21.833

Temuan: EDA terbukti lebih stabil pada UC-01 karena secara konsisten menghasilkan throughput lebih tinggi, latensi end-to-end lebih rendah, serta menekan error hingga 0,00% pada seluruh level beban.

2) Analisis UC-02 (Stock Adjustment)

TABEL V
HASIL STRESS TEST UC-02 (STOCK ADJUSTMENT)

Beban	Arsitektur	Throughput	Latensi (E2E)	Error (%)	Request Dropped
50 RPS	Monolith	18,76 req/s	21,16 s	16,61%	3.709
	EDA	18,84 req/s	21,73 s	28,06%	3.613
100 RPS	Monolith	20,95 req/s	34,80 s	21,54%	8.721
	EDA	19,25 req/s	37,62 s	30,74%	8.762
200 RPS	Monolith	22,85 req/s	57,06 s	25,40%	25.299
	EDA	20,67 req/s	62,00 s	35,99%	25.403

Temuan: Modular monolith relatif lebih baik pada UC-02. Penambahan queue, serialisasi, dan pemrosesan antrian tambahan pada EDA justru menambah overhead sehingga bottleneck akibat database locking menjadi semakin terlihat saat beban meningkat.

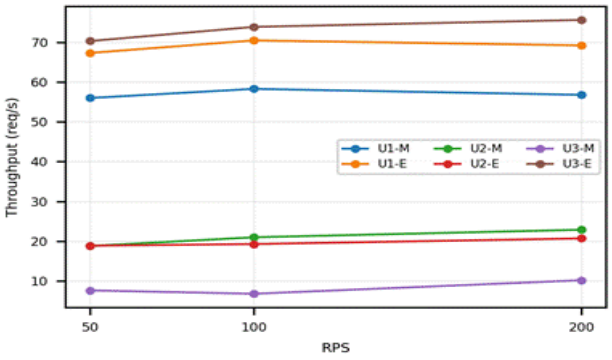
Akar permasalahan pada UC-02 terletak pada mekanisme *pessimistic locking* dalam proses pembaruan stok, yang bekerja dengan mengunci baris data di basis data sejak transaksi dimulai hingga selesai demi mencegah inkonsistensi data akibat pembaruan ganda. Meskipun pendekatan ini tepat untuk menjaga integritas data, penerapannya pada arsitektur EDA justru memperburuk kondisi karena menimbulkan antrean eksekusi di level basis data. Di sini, *worker* yang memproses *event* secara paralel turut berebut kunci yang sama, sehingga kontesi penguncian (*lock contention*) semakin meningkat seiring bertambahnya beban. Dengan demikian, pemindahan proses ke antrean tidak mampu mengurangi *bottleneck* karena titik hambat utamanya bukan terletak pada beban komputasi, melainkan pada penguncian baris basis data yang bersifat serial.

3) Analisis UC-03 (Outbound Confirmation)

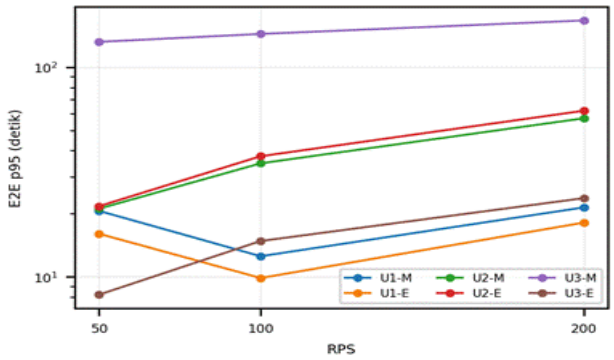
TABEL VI
HASIL STRESS TEST UC-03 (OUTBOUND CONFIRMATION)

Beban	Arsitektur	Throughput	Latensi (E2E)	HTTP Error	Gagal Fungsi
50 RPS	Monolith	7,60 req/s	132,00 s	58,05%	53,82%
	EDA	70,32 req/s	8,23 s	0,00%	0,00%
100 RPS	Monolith	6,74 req/s	144,00 s	53,74%	52,69%
	EDA	73,89 req/s	14,83 s	0,00%	0,00%
200 RPS	Monolith	10,14 req/s	167,00 s	58,76%	81,00%
	EDA	75,64 req/s	23,74 s	0,00%	0,00%

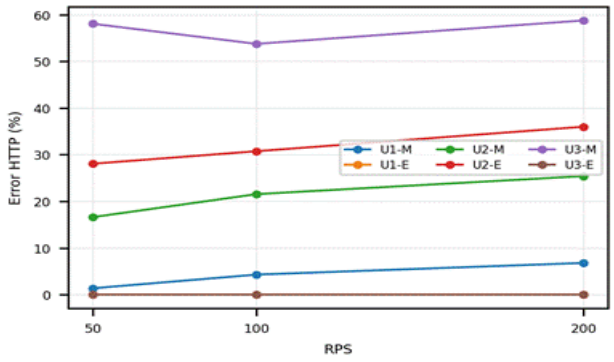
Temuan: EDA unggul mutlak pada UC-03. Pemisahan proses notifikasi dan side effect dari jalur request utama mencegah timeout total, mempertahankan error 0,00%, dan menjaga fungsi sistem tetap berjalan pada seluruh fase beban.



Gbr. 3 Tren throughput aktual pada stress test 50, 100, dan 200 RPS.



Gbr. 4 Tren latensi p(95) end-to-end pada stress test 50, 100, dan 200 RPS. Sumbu y menggunakan skala logaritmik.



Gbr. 5 Tren error rate HTTP pada stress test 50, 100, dan 200 RPS.

Temuan lintas fase menunjukkan tiga pola utama. Pertama, UC-01 memperoleh manfaat stabil dari EDA karena proses pendukung dapat dipindahkan ke queue tanpa mengganggu transaksi utama. Kedua, UC-02 tetap dibatasi oleh row-level lock sehingga pemrosesan asinkron justru menambah overhead. Ketiga, UC-03 merupakan skenario yang paling cocok untuk EDA karena proses notifikasi dan side effect dapat dipisahkan dari jalur respons.

C. *Hasil Pengujian Stress Test*

Secara keseluruhan, hasil pengujian menunjukkan bahwa EDA bukan solusi universal, melainkan efektif ketika bottleneck berasal dari proses pendukung yang dapat dipisahkan dari jalur request utama. UC-01 dan UC-03 memperoleh manfaat karena logging, notifikasi, dan pembaruan dashboard dapat didelegasikan ke worker. Sebaliknya, UC-02 menunjukkan bahwa EDA tidak dapat memperbaiki performa apabila titik hambatnya berada pada operasi basis data yang tetap serial dan saling mengunci.

Implikasi praktisnya adalah penerapan EDA pada WMS sebaiknya dilakukan secara selektif. Modul dengan alur bisnis panjang, banyak side effect, dan proses non-kritis cocok dipindahkan ke queue. Modul yang sangat bergantung pada konsistensi stok dan transaksi database perlu terlebih dahulu diperbaiki melalui strategi mekanisme penguncian basis data yang lebih efisien sebelum di delegasikan ke worker.

IV. KESIMPULAN DAN SARAN

Penelitian ini menyimpulkan bahwa penerapan Event-Driven Architecture secara inkremental dapat meningkatkan kinerja dan skalabilitas aplikasi gudang berbasis web, terutama pada use case dengan proses pendukung yang dapat dipisahkan dari jalur transaksi utama. Pada *outbound*, EDA menurunkan latensi beban ringan dari 18,84 detik menjadi 204,00 ms dan meningkatkan throughput stress test dari 10,14 req/s menjadi 75,64 req/s. Pada *inbound*, EDA juga meningkatkan stabilitas dengan error rate 0,00% pada stress test. Namun, pada *adjustment*, EDA belum optimal karena overhead queue dan mekanisme antrian memperburuk bottleneck database lock yang masih menggunakan pessimistic locking.

Berdasarkan hasil tersebut, penelitian lanjutan disarankan menerapkan pola asinkron yang lebih menyeluruh secara lebih penuh, mengganti pessimistic locking dengan optimistic locking berbasis versioning agar pemrosesan event lebih andal. Selain itu, pengujian berikutnya perlu memperluas variasi dataset, durasi beban, jumlah worker, dan konfigurasi resource agar dampak scaling EDA dapat dievaluasi secara lebih menyeluruh.

REFERENSI

[1] N. Andiyappillai, "Digital Transformation in Warehouse Management Systems (WMS) Implementations," International Journal of Computer Applications, vol. 177, no. 45, hal. 34-37, 2020.
[2] C. Aguedo, J. Espinoza, dan A. Pacheco, "Improving Inventory Control Through a Web-Based System in a Retail Company," F1000Research, vol. 13, hal. 252, 2024.
[3] R. Setiawan, N. P. Sugihartanti, dan M. I. Ibadurrahman, "Sistem Manajemen Gudang Berbasis Web dengan Teknologi Barcode Scanner pada Industri Manufaktur: Sebuah Kajian Literatur," 2024.
[4] M. A. Maulidi dan A. S. W. Prasetyawati, "Design of a Web-Based Inventory Management System For Toko Fajar Mandiri Tani In Karangmulya, Suradadi," Journal of Artificial Intelligence and Engineering Applications (JAIEA), vol. 4, no. 2, hal. 618-622, 2025.
[5] K. Shanmugamani dan F. Mohamad, "The Implementation of Warehouse Management System (WMS) to Improve Warehouse Performance in Business to Business (B2B)," International Journal of Industrial Management, vol. 17, no. 4, hal. 231-239, 2023.
[6] Z. Romadhon, "Inventory Management System Framework Based on Python and MySQL Database with Blockchain Technology Implementation," vol. 5, no. 3, 2025.
[7] S. Yerra, "Enhancing Inventory Management through Real-Time Power BI Dashboards and KPI Tracking," International Journal of Scientific Research in Computer Science, Engineering and Information Technology, vol. 11, no. 2, hal. 944-951, 2025.
[8] C. L. Zavaleta Saenz, A. Romero Ruiz, dan A. Pacheco, "Web/Mobile system innovation: An efficient revolution in warehouse management," F1000Research, vol. 13, hal. 213, 2024.
[9] G. Blinowski, A. Ojdowska, dan A. Przybylek, "Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation," IEEE Access, vol. 10, hal. 20357-20374, 2022.
[10] L. F. Al-Qora'n dan A. Al-Said Ahmad, "Modular Monolith Architecture in Cloud Environments: A Systematic Literature Review," Future Internet, vol. 17, no. 11, hal. 496, 2025.
[11] L. Krzyszton, K. Latwinski, dan M. Plechawska-Wojcik, "Comparative analysis of selected aspects of web application architectures," Journal of Computer Sciences Institute, vol. 34, hal. 81-88, 2025.
[12] D. Faustino, N. Goncalves, M. Portela, dan A. R. Silva, "Stepwise Migration of a Monolith to a Microservices Architecture: Performance and Migration Effort Evaluation," arXiv:2201.07226, 2022.
[13] S. Tragatschnig, S. Stevanetic, dan U. Zdun, "Supporting the evolution of event-driven service-oriented architectures using change patterns," Information and Software Technology, vol. 100, hal. 133-146, 2018.

[14] S. Khriji, Y. Benbelgacem, R. Cheour, D. E. Houssaini, dan O. Kanoun, "Design and implementation of a cloud-based event-driven architecture for real-time data processing in wireless sensor networks," The Journal of Supercomputing, vol. 78, no. 3, hal. 3374-3401, 2022.

[15] A. Dhanaraj, "Building Resilient Systems: Error Handling, Retry Mechanisms, and Predictive Analytics in Event-Driven Architecture."

[16] L. R. V. -, "Optimizing Scalability and Decoupling with Event-Driven Architecture: A Cross-Industry Analysis and A Comparative Perspective," International Journal on Science and Technology, vol. 16, no. 1, hal. 2190, 2025.

[17] A. Anwar, "Event-Driven Architecture in Distributed Systems: Leveraging Azure Cloud Services for Scalable Applications," European Journal of Computer Science and Information Technology, vol. 13, no. 29, hal. 13-27, 2025.

[18] R. V. Bhupatiraju, "Event-Driven Architecture for Payment Failover and Redundancy: A Framework for High-Availability Financial Transaction Processing," European Modern Studies Journal, vol. 9, no. 5, hal. 815-830, 2025.

[19] H. Alnuhait dkk., "Web application performance assessment: A study of responsiveness, throughput, and scalability," International Journal of ADVANCED AND APPLIED SCIENCES, vol. 11, no. 9, hal. 214-226, 2024.

[20] Q. Fan dan Q. Wang, "Performance Comparison of Web Servers with Different Architectures: A Case Study Using High Concurrency Workload," 2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb), hal. 37-42, 2015.