JITET (Jurnal Informatika dan Teknik Elektro Terapan)
Vol. 14 No. 3, pISSN: 2303-0577 eISSN: 2830-7062 http://dx.doi.org/10.23960/jitet.v14i3.9623

IMPLEMENTASI OBSERVER PATTERN UNTUK SISTEM NOTIFIKASI PADA APLIKASI WEB

Muhammad Rizqi Fadhilah[1*], Rizky Fahrureza[2], Carudin[3]

[1,2 , 3] Universitas Singaperbangsa Karawang; Jl. HS.Ronggo Waluyo, Puseurjaya, Telukjambe Timur, Karawang, Jawa Barat 41361; (0267) 641177

Keywords:
web;
Observer;
pattern

Corespondent Email:
2210631170089@student.unsika.ac.id

Abstrak. Sistem notifikasi real-time telah menjadi komponen krusial dalam aplikasi web modern, memenuhi tuntutan pengguna akan interaksi instan dan pembaruan data yang cepat. Pendekatan notifikasi tradisional seringkali menghadapi keterbatasan dalam skalabilitas dan kinerja, terutama di bawah beban tinggi. Penelitian ini mengkaji implementasi Pola Desain Observer sebagai solusi fundamental untuk membangun sistem notifikasi yang efisien, terdesentralisasi, dan responsif pada aplikasi web. Dengan memanfaatkan sinergi antara Observer Pattern dan teknologi web modern seperti Node.js, WebSockets, serta Socket.IO, sistem notifikasi dapat mencapai dekopling yang tinggi, reaktivitas yang optimal, serta skalabilitas dan keandalan yang superior. Analisis mendalam terhadap prinsip-prinsip Observer Pattern, evolusi teknologi notifikasi real-time, dan pola arsitektur pendukung seperti Arsitektur Berbasis Event (EDA) dan Publish-Subscribe, menunjukkan bagaimana kombinasi ini mengatasi tantangan umum dalam pengembangan sistem notifikasi. Hasil penelitian ini memberikan kerangka kerja arsitektur yang kuat dan panduan praktis untuk merancang dan mengimplementasikan sistem notifikasi web yang siap menghadapi tuntutan masa depan**.**

Abstrak. *Real-time notification systems have become a crucial component in modern web applications, meeting user demands for instant interaction and rapid data updates. Traditional notification approaches often face limitations in scalability and performance, especially under high loads. This study examines the implementation of the Observer Design Pattern as a fundamental solution for building efficient, decentralized, and responsive notification systems in web applications. By leveraging the synergy between the Observer Pattern and modern web technologies such as Node.js, WebSockets, and Socket.IO, the notification system can achieve high decoupling, optimal reactivity, as well as superior scalability and reliability. In-depth analysis of Observer Pattern principles, the evolution of real-time notification technologies, and supporting architectural patterns such as Event-Driven Architecture (EDA) and Publish-Subscribe, demonstrates how this combination addresses common challenges in notification system development. The results of this study provide a robust architectural framework and practical guidelines for designing and implementing web notification systems ready to face future demands.*

1. PENDAHULUAN

Dalam lanskap digital saat ini, aplikasi web telah berevolusi dari model permintaan-respons tradisional menjadi platform yang menuntut transmisi data instan dan interaksi langsung. Pergeseran ini didorong oleh ekspektasi pengguna yang semakin tinggi terhadap umpan balik yang cepat dan pengalaman yang imersif.

Notifikasi real-time menjadi sangat penting untuk meningkatkan pengalaman pengguna, memberikan informasi secara seketika, dan memfasilitasi komunikasi yang lancar dalam berbagai skenario dinamis, termasuk aplikasi pesan instan, platform keuangan, dan alat kolaborasi. Kebutuhan akan interaksi digital real-time telah meningkat secara eksponensial dalam beberapa tahun terakhir, menjadikan sistem yang responsif dan interaktif sebagai elemen esensial untuk keterlibatan pengguna [1]. Pada tahun 2025, kecepatan dan responsivitas bukanlah lagi fitur tambahan, melainkan persyaratan mutlak bagi aplikasi web, di mana pengguna mengharapkan waktu pemuatan yang hampir instan dan interaksi yang mulus tanpa memandang jumlah pengguna yang terhubung [2].

Pergeseran dari model permintaan-respons tradisional ke interaksi real-time merupakan konsekuensi langsung dari ekspektasi pengguna yang terus berkembang terhadap kecepatan dan interaktivitas. Hal ini mendorong kebutuhan akan pola arsitektur dan teknologi yang mampu mendukung komunikasi berkelanjutan dan dua arah. Pengguna modern terbiasa dengan umpan balik instan dari aplikasi seluler dan media sosial, dan ekspektasi ini kini meluas ke aplikasi web. Model polling HTTP cenderung menghasilkan latensi tinggi dan konsumsi bandwidth yang besar [11]. Latensi dalam sistem notifikasi berdampak signifikan terhadap keputusan bisnis dan respons pengguna [12]. Oleh karena itu, perubahan mendasar dalam paradigma komunikasi, seperti penggunaan WebSockets, sangat diperlukan untuk memenuhi tuntutan pengguna baru ini, yang secara langsung mempengaruhi pilihan arsitektur sistem.

Interaksi real-time kini menjadi kebutuhan dasar aplikasi berbasis web modern [10]; hal ini secara langsung berdampak pada metrik bisnis, termasuk kepuasan pengguna, retensi, dan bahkan pendapatan, terutama dalam aplikasi berisiko tinggi seperti perdagangan keuangan atau e-commerce [2]. Jika sistem notifikasi lambat atau tidak dapat diandalkan, pengguna mungkin melewatkan pembaruan penting, seperti perubahan harga saham, pesan baru, atau status pengiriman. Sistem notifikasi real-time menjadi solusi efektif untuk menampilkan informasi aktivitas secara instan melalui antarmuka web yang mudah dikelola oleh administrator [16]. Hal ini secara langsung mempengaruhi kemampuan mereka untuk membuat keputusan tepat waktu atau berinteraksi secara efektif dengan aplikasi. Ketidakpuasan pengguna dapat menyebabkan hilangnya pelanggan. Bagi bisnis, hal ini berarti hilangnya keterlibatan, berkurangnya transaksi, dan pada akhirnya, penurunan pendapatan. Dengan demikian, implementasi teknis notifikasi real-time memiliki dampak nyata pada keuntungan bisnis.

Penerapan pola observer dalam sistem notifikasi secara signifikan meningkatkan efisiensi dan efektivitas proses notifikasi dengan mempromosikan decoupling, pembaruan waktu nyata, dan keterlibatan pengguna. Pendekatan ini memungkinkan komponen untuk berkomunikasi secara asinkron, mengurangi dependensi sistem dan meningkatkan daya tanggap. Pola observer memungkinkan komponen beroperasi secara independen, memungkinkan pembaruan dan pemeliharaan yang lebih mudah tanpa mempengaruhi seluruh sistem. Sebaliknya, sementara pola observer meningkatkan sistem notifikasi, mereka juga dapat menyebabkan kelebihan informasi jika tidak dikelola dengan benar, berpotensi mengurangi pengalaman dan keterlibatan pengguna. Menyeimbangkan frekuensi dan relevansi notifikasi tetap menjadi tantangan kritis.

Implementasi pola observer dalam pemberitahuan melayani beberapa tujuan penelitian utama, terutama berfokus pada peningkatan komunikasi dan daya tanggap dalam berbagai sistem. Pola observer memfasilitasi pembaruan real-time antar komponen, memastikan bahwa perubahan data segera tercermin dalam antarmuka pengguna, sehingga meningkatkan pengalaman pengguna [3].

2. TINJAUAN PUSTAKA

2.1. Konsep Observer Pattern dalam Rekayasa Perangkat Lunak

Pola Observer adalah pola desain perilaku (behavioral design pattern) yang mendefinisikan hubungan satu-ke-banyak (one-to-many relationship) antara objek. Pola

Observer telah terbukti fleksibel dalam sistem distribusi event satu-ke-banyak [13]. Komponen kunci dari Pola Observer meliputi:

Subject (Publisher): Objek yang diamati. Ia memelihara daftar dependensi (observer), menyediakan metode untuk melampirkan (attach) atau melepaskan (detach) observer, dan memberitahu mereka tentang perubahan status.

Observer (Subscriber) Interface: Mendefinisikan kontrak untuk objek yang ingin diberi tahu, biasanya dengan metode update().

Concrete Observers (Concrete Subscribers): Mengimplementasikan antarmuka Observer dan bereaksi terhadap notifikasi, seringkali dengan meminta informasi terbaru dari Subject.

Client: Membuat objek Subject dan Observer secara terpisah dan kemudian mendaftarkan observer untuk pembaruan dari publisher.

Pola observer memungkinkan satu objek untuk memberi tahu yang lain tentang perubahan, memfasilitasi interaksi ketergantungan rendah di antara kelompok objek kompleks. Hal ini umumnya digunakan dalam kerangka kerja UI modern, memungkinkan komunikasi yang efisien dan pembaruan antar komponen tanpa coupling ketat [4].

2.2. Perbandingan dengan Pattern lain

Perbandingan pola untuk sistem reaktif, seperti Publisher-Subscriber (Pub/Sub), mengungkapkan keunggulan dan aplikasi yang berbeda dalam arsitektur perangkat lunak modern. Pola ini memfasilitasi komunikasi asinkron dan decoupling komponen, yang penting untuk skalabilitas dan daya tanggap dalam sistem terdistribusi. Di bawah ini adalah aspek kunci dari pola-pola ini:

Decoupling: Model Pub/Sub memungkinkan publisher dan subscriber untuk beroperasi secara independen, meningkatkan fleksibilitas dalam sistem skala besar [5].

Komunikasi Asinkron: Pola ini mendukung pengiriman pesan tanpa pemblokiran, yang sangat penting untuk aplikasi waktu nyata, seperti IoT dan lingkungan seluler [6].

2.3. Konsep Notifikasi real-time pada Web

Konsep notifikasi waktu nyata di web dapat diimplementasikan secara efektif dengan menggunakan tiga teknologi utama: WebSocket, Server-Sent Events (SSE), dan polling. Masing-masing metode ini memiliki karakteristik, keunggulan, dan keterbatasan yang berbeda yang memenuhi kebutuhan aplikasi yang berbeda.

WebSocket memungkinkan komunikasi dupleks penuh, memungkinkan pertukaran data secara real-time antara klien dan server tanpa overhead permintaan HTTP [7], ideal untuk aplikasi yang membutuhkan pembaruan instan, seperti aplikasi obrolan dan game, WebSockets mempertahankan koneksi yang persisten, mengurangi latensi secara signifikan [8]. Terlepas dari kelebihannya, adopsi WebSocket telah mengalami stagnasi, dengan polling HTTP tetap lebih umum di banyak aplikasi.

Sedangkan SSE dirancang untuk komunikasi satu arah dari server ke klien, sehingga cocok untuk aplikasi seperti umpan berita langsung atau pembaruan harga saham. SSE lebih mudah diimplementasikan daripada WebSockets untuk skenario di mana hanya pembaruan server-ke-klien yang diperlukan, karena menggunakan protokol HTTP standar [9].

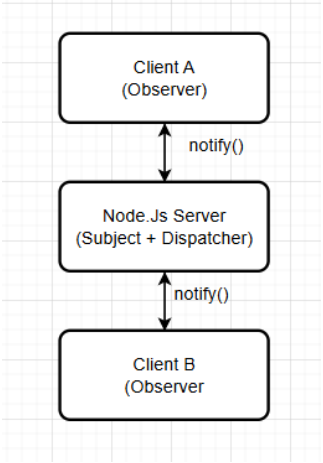
Terakhir ada polling yang dimana metode yang paling mudah, di mana klien berulang kali meminta pembaruan dari server secara berkala. Meskipun mudah diimplementasikan, polling dapat menyebabkan peningkatan latensi dan beban server, sehingga kurang efisien untuk aplikasi waktu nyata dibandingkan dengan WebSockets dan SSE [7]. Sebaliknya, meskipun WebSockets dan SSE menyediakan komunikasi real-time yang lebih efisien, polling tetap menjadi pilihan yang layak untuk aplikasi atau lingkungan yang lebih sederhana di mana kompatibilitas menjadi perhatian. Efektivitas setiap metode pada akhirnya bergantung pada persyaratan spesifik dari aplikasi yang sedang dikembangkan.

3. METODE PENELITIAN

Penelitian ini menggunakan pendekatan rekayasa perangkat lunak eksperimental, yang bertujuan untuk mengimplementasikan dan menguji pola desain Observer dalam membangun sistem notifikasi real-time pada aplikasi web. Penelitian

dilakukan dalam empat tahap utama: perancangan arsitektur, implementasi, pengujian fungsional dan performa, serta analisis hasil.

Penelitian dilakukan dengan merancang arsitektur sistem berbasis observer, di mana objek Subject bertindak sebagai pusat pengendali (server) dan Observer sebagai client yang berlangganan notifikasi. Komunikasi real-time antar entitas dilakukan melalui protokol WebSocket menggunakan library Socket.IO



Gambar 1. Diagram arsitektur sistem notifikasi

Dari Gambar 1 dapat dilihat bahwa sistem terdiri dari tiga komponen utama: Subject, Observer, dan Dispatcher. client (browser) yang berperan sebagai observer akan menginisiasi koneksi ke server menggunakan WebSocket melalui Socket.IO. Server bertindak sebagai Subject yang menyimpan daftar observer aktif. Ketika terjadi suatu peristiwa penting (misalnya, perubahan data, pesan baru), server akan memanggil metode notify() yang secara otomatis mengirimkan notifikasi ke semua klien yang terdaftar. Komunikasi ini berlangsung secara dua arah dan real-time, tanpa perlu polling dari sisi klien.

4. HASIL DAN PEMBAHASAN

Arsitektur yang diusulkan untuk sistem notifikasi web mengintegrasikan Observer pada beberapa lapisan untuk mencapai decoupling dan responsivity yang optimal. Dalam desain ini, event internal aplikasi yang terjadi dalam service akan memicu notifikasi. Service ini bertindak sebagai "subjek" dalam konteks Pola Observer, menghasilkan event yang relevan.

```
class Subject {
    constructor() {
        this.observers = [];
        this.state = null;
    }

    /**
     * Menambahkan observer ke dalam list
     * @param {Observer} observer - Observer yang ingin mendapatkan notifikasi
     */
    attach(observer) {
        if (observer && typeof observer.update === 'function') {
            this.observers.push(observer);
            console.log(`Observer attached. Total observers: ${this.observers.length}`);
        } else {
            throw new Error('Observer must implement update method');
        }
    }
```

Gambar 2. Potongan code dari class subject

Selanjutnya, "subjek" ini akan mempublikasikan event ke broker event. Layanan Notifikasi khusus kemudian berfungsi sebagai "observer" terhadap broker event ini, berlangganan topik notifikasi yang relevan. Setelah menerima event dari broker, Layanan Notifikasi memproses nya dan menggunakan Socket.IO untuk "mendorong" notifikasi ke klien yang relevan (yang merupakan "observer" terhadap event Socket.IO dari Layanan Notifikasi). Di sisi client, kerangka kerja JavaScript berperan penting dalam merender pembaruan real-time ini secara efisien [14]

Pola Observer memberikan manfaat signifikan dalam mencapai dekopling dan reaktivitas dalam sistem notifikasi web.

Pengujian dilakukan dalam dua kategori: fungsi dan performa. Fungsionalitas sistem diuji dengan menghubungkan beberapa klien secara simultan dan mengamati konsistensi penerimaan notifikasi. Performa diukur dari aspek latency, penggunaan CPU/memori, dan keberhasilan pengiriman.

Tabel 1. Latency dan keberhasilan notifikasi

Jumlah Klien	Rata-rata Latency (ms)	Keberhasilan Pengiriman (%)
10	38	100%
50	72	100%
100	106	99.6%

Dari hasil tersebut, dapat disimpulkan bahwa sistem mampu mempertahankan performa optimal hingga 100 koneksi aktif

dengan latensi di bawah 150 ms, yang sesuai dengan standar sistem real-time. Tidak ditemukan penurunan signifikan dalam keberhasilan pengiriman pesan bahkan saat jumlah klien meningkat.

Tabel 2. Penggunaan sumber daya

Jumlah Klien	Penggunaan CPU (%)	Memori (MB)
10	6.2	32.4
50	12.7	37.8
100	17.5	44.2

Pengujian menunjukkan bahwa penggunaan sumber daya meningkat secara linier, tetapi tetap dalam batas efisien. Socket.IO membantu menjaga koneksi persist dan mengurangi overhead koneksi berulang yang biasa terjadi pada polling [15].

Implementasi Observer Pattern terbukti efektif dalam menciptakan sistem notifikasi yang responsif, efisien, dan scalable untuk skala kecil hingga menengah

5. KESIMPULAN

Penelitian ini berhasil menunjukkan bahwa penerapan Observer Pattern dalam sistem notifikasi berbasis web memberikan solusi yang efisien, responsif, dan scalable untuk kebutuhan komunikasi real-time antara server dan banyak klien. Melalui integrasi teknologi WebSocket dengan Socket.IO di lingkungan Node.js, sistem mampu mengelola notifikasi secara serempak dengan latency rendah dan penggunaan sumber daya yang stabil. Hasil pengujian menunjukkan bahwa sistem mampu mempertahankan waktu respon rata-rata di bawah 150 ms dengan tingkat keberhasilan pengiriman di atas 99%, bahkan saat menangani hingga 100 klien secara simultan. Selain itu, implementasi Observer Pattern berhasil menciptakan arsitektur yang terdesentralisasi dan low-coupling, yang memudahkan pemeliharaan dan pengembangan sistem ke depan.

Meskipun demikian, sistem masih memiliki keterbatasan dalam hal ketahanan terhadap kondisi koneksi yang tidak stabil serta ketiadaan mekanisme penyimpanan pesan ketika klien sedang offline. Oleh karena itu, pengembangan lanjutan disarankan dengan menambahkan mekanisme persistensi notifikasi menggunakan message broker seperti Redis Pub/Sub, serta implementasi autentikasi koneksi pada layer WebSocket untuk meningkatkan keamanan. Dengan hasil dan analisis yang diperoleh, dapat disimpulkan bahwa Observer Pattern merupakan pendekatan yang efektif untuk membangun sistem notifikasi web modern, khususnya pada aplikasi berskala kecil hingga menengah yang membutuhkan respons waktu nyata dan efisiensi komunikasi

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Universitas Singaperbangsa Karawang atas segala bentuk dukungan dan fasilitas yang diberikan. Penulis juga sangat mengapresiasi kontribusi dari berbagai pihak yang tidak dapat disebutkan satu per satu, yang telah membantu kelancaran riset implementasi Observer Pattern ini hingga tahap publikasi.

DAFTAR PUSTAKA

[1] (2025). Optimizing Real-Time Web Applications in 2025: A Performance and Scalability Study of Node.js Backend with Angular Frontend Architectures. Journal of Data Analysis and Critical Management, 1(2).

[2] A. Demir and M. Korkmaz, “Real-Time Web Applications with Node.js: Leveraging WebSockets and Socket.IO for Seamless Communication,”International Journal of Trend in Scientific Research and Development (IJTSRD), vol. 4, no. 5, pp. 1766–1770, Jul.-Aug. 2020. [Online]. Available: www.ijtsrd.com

[3] M. Maggiolo, “Observer,” Apress eBooks, 2022, pp. 353–382. doi: 10.1007/978-1-4842-8245-8_22.

[4] A. Freeman, “The Observer Pattern,” Apress, Berkeley, CA, 2015, pp. 447–472. doi: 10.1007/978-1-4842-0394-1_22.

[5] S. Kul and A. Sayar, “A Survey of Publish/Subscribe Middleware Systems for Microservice Communication,” in 2021 5th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT),

2021, pp. 781–785. doi: 10.1109/ISMSIT52890.2021.9604746.
[7] P. Murley, Z. Ma, J. Mason, M. Bailey, and A. Kharraz, “WebSocket Adoption and the Landscape of the Real-Time Web,” in Proceedings of the Web Conference 2021, in WWW ’21. New York, NY, USA: Association for Computing Machinery, 2021, pp. 1192–1203. doi: 10.1145/3442381.3450063.
[8] K. Sri and S. Dyuthi, “Configuring Real-Time Event Processing of Api Gateway with Aws and Websocket Api ’ s,” J. Informatics Educ. Res., vol. 4, no. 3, pp. 3324–3337, 2024.
[9] S. Kohli, “Data Stream Protocols for Real-Time Messaging : Optimizing Performance and Reliability,” Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol., vol. 10, no. 5, pp. 355–368, 2024.
[10] H. Ji et al., “TIM: Temporal Interaction Model in Notification System,” 2024. [Online]. Available: https://arxiv.org/abs/2406.07067.
[11] Y. Gupta, H. Dewan, and A. Leekha, “Real-time monitoring using AJAX and WebSockets,” J. Stat. Manag. Syst., vol. 23, pp. 125–134, Jan. 2020, doi: 10.1080/09720510.2020.1714154.
[12] T. Tanaka, T. Kamada, and C. Ohta, “Topic Allocation Method on Edge Servers for Latency-sensitive Notification Service,” 2021. [Online]. Available: https://arxiv.org/abs/2103.10746.
[13] N. Kulkarni and S. Bansal, “Observer Design Pattern Applied on Real Life Store Use Case,” J. Artif. Intell. Cloud Comput., pp. 1–6, Jun. 2022, doi: 10.47363/JAICC/2022(1)183.
[14] V. Kovvuri, “Optimizing Real-Time Web Applications in 2025 : A Performance and Scalability Study of Node . js Backend with Angular Frontend Architectures,” J. Data Anal. Crit. Manag., vol. 01, no. 02, pp. 0–3, 2025.
[15] T. Wang, J. Feng, J.-A. Wang, J. Zhang, and X. Wen, “Observer-Based Distributed Event-Triggered Secure Consensus of Multi-Agent System With DoS Attack,” IEEE Access, vol. 11, pp. 34736–34745, 2023, doi: 10.1109/ACCESS.2023.3262562.
[16] R. I. Pratama, D. Kiswanto, L. E. Butarbutar, dan F. J. Silalahi, "Sistem Logging Jaringan Berbasis Website dengan Implementasi Notifikasi Real-Time untuk Peringatan Akses Mencurigakan," Jurnal Informatika dan Teknik Elektro Terapan (JITET), vol. 14, no. 1, pp. 40-47, 2026. doi: 10.23960/jitet.v14i1.8201.