E -ISSN : 2746-0835
Volume 6 No 1 (2025) JUSTI (Jurnal Sistem Dan Teknik Industri)

PENERAPAN PROTOKOL WEBSOCKET UNTUK SISTEM NOTIFIKASI PERUBAHAN DATA (LIVE UPDATE) PADA APLIKASI WEB E-COMMERCE

Hafidz irsyad Habibullah[1], Henny Dwi Bhakti[2]
Program Studi Teknik Informatika Fakultas Teknik, Universitas Muhammadiyah Gresik
Jl. Sumatera 101 GKB, Gresik 61121, Indonesia
e-mail : hfidzsans@gmail.com

ABSTRAK

Aplikasi e-commerce modern menuntut interaksi data yang cepat dan *real-time* untuk meningkatkan pengalaman pengguna (UX). Informasi krusial seperti perubahan stok barang, status pesanan, dan notifikasi *flash sale* seringkali tertinggal karena arsitektur *request-response* HTTP tradisional yang mengharuskan pengguna melakukan *refresh* halaman. Metode alternatif seperti AJAX *polling* (short/long) dapat membebani server dan meningkatkan latensi. Penelitian ini bertujuan merancang dan mengimplementasikan sistem notifikasi *live update* pada aplikasi web e-commerce menggunakan protokol WebSocket. WebSocket menyediakan saluran komunikasi *full-duplex* (dua arah) yang persisten melalui satu koneksi TCP, memungkinkan server mengirimkan data ke klien secara proaktif. Metode penelitian yang digunakan adalah *prototyping*, dengan implementasi sistem menggunakan Node.js (dengan *library* Socket.IO) sebagai server WebSocket dan JavaScript murni di sisi klien. Pengujian fungsional menunjukkan bahwa sistem berhasil mengirimkan notifikasi perubahan stok dan status pesanan secara instan ke dasbor pengguna dan halaman produk. Hasil pengujian kinerja menunjukkan bahwa WebSocket secara signifikan mengurangi latensi pengiriman data (rata-rata di bawah 400ms) dibandingkan dengan AJAX *short-polling* (rata-rata 3000ms dengan interval 3 detik) dan mengurangi *overhead* koneksi HTTP pada server. Disimpulkan bahwa WebSocket adalah solusi yang sangat efektif dan efisien untuk kebutuhan *live update* data pada platform e-commerce.

Kata kunci : E-Commerce, Live Update, Notifikasi Real-Time, Protokol WebSocket, Socket.IO.

ABSTRACT

Modern e-commerce applications demand fast and real-time data interaction to enhance the user experience (UX). Crucial information, such as stock level changes, order statuses, and flash sale notifications, often lags due to the traditional HTTP request-response architecture, which requires users to refresh the page. Alternative methods like AJAX polling (short/long) can overload the server and increase latency. This research aims to design and implement a live update notification system for an e-commerce web application using the WebSocket protocol. WebSocket provides a persistent, full-duplex (two-way) communication channel over a single TCP connection, enabling the server to proactively send data to the client. The research method used is prototyping, with the system implemented using Node.js (with the Socket.IO library) as the WebSocket server and pure JavaScript on the client side. Functional testing shows that the system successfully delivered instant notifications for stock changes and order statuses to the user dashboard and product pages. Performance testing results indicate that WebSocket significantly reduces data delivery latency (average under 400ms) compared to AJAX short-polling (average 3000ms with a 3-second interval) and reduces HTTP connection overhead on the server. It is concluded that WebSocket is a highly effective and efficient solution for the live update data needs of e-commerce platforms

***Keywords** : E-Commerce, Live Update, Real-Time Notifications, WebSocket Protocol, Socket.IO.*

Jejak Artikel
Upload artikel : 12 Oktober 2025
Revisi : 25 Oktober 2025
Publish : 29 Oktober 2025

1. PENDAHULUAN

Persaingan di industri e-commerce sangat bergantung pada kepuasan dan pengalaman

pengguna (UX). Salah satu aspek penting dari UX modern adalah penyajian data yang *real-time*. Pengguna mengharapkan informasi yang mereka lihat di layar selalu akurat, terutama data yang sensitif terhadap waktu seperti ketersediaan stok barang, harga *flash sale*, atau pembaruan status pengiriman pesanan [1].

Secara tradisional, aplikasi web dibangun di atas protokol HTTP yang bersifat *stateless* dan *request-response*. Artinya, server tidak dapat mengirim data ke klien (browser) kecuali klien memintanya terlebih dahulu. Untuk mengatasi keterbatasan ini, pengembang menggunakan teknik seperti AJAX *polling* [2]. Terdapat dua jenis *polling*:

1. **Short Polling:** Klien secara periodik (misalnya, setiap 5 detik) mengirim permintaan ke server untuk menanyakan "apakah ada data baru?". Metode ini tidak efisien, menciptakan banyak *traffic* HTTP yang tidak perlu dan data yang diterima tidak benar-benar *real-time* (ada jeda hingga 5 detik).
2. **Long Polling:** Klien mengirim permintaan, tetapi server menahan koneksi tersebut tetap terbuka sampai ada data baru untuk dikirim. Meskipun lebih baik dari *short polling*, metode ini masih membebani server karena harus mengelola banyak koneksi HTTP yang menggantung [3].

Protokol WebSocket (RFC 6455) diperkenalkan untuk mengatasi masalah ini. WebSocket menginisiasi koneksi melalui *handshake* HTTP, kemudian "meningkatkan" koneksi tersebut menjadi koneksi TCP *full-duplex* yang persisten [4]. Setelah koneksi terjalin, server dan klien dapat saling mengirim data kapan saja tanpa perlu melakukan permintaan HTTP baru. Hal ini memungkinkan server untuk secara proaktif "mendorong" (*push*) data ke klien begitu peristiwa terjadi.

Penelitian ini mengusulkan penerapan protokol WebSocket untuk membangun sistem notifikasi *live update* yang fokus pada dua fitur kritis e-commerce: perubahan ketersediaan stok produk dan pembaruan status pesanan. Tujuan utamanya adalah untuk membuktikan bahwa WebSocket dapat memberikan pembaruan yang lebih cepat dan lebih efisien (dari segi sumber daya server) dibandingkan dengan metode *polling* tradisional.

2. METODOLOGI PENELITIAN

Metodologi yang digunakan dalam penelitian ini adalah model *prototyping*. Tahapan penelitian mencakup analisis kebutuhan, perancangan arsitektur, implementasi *prototype*, pengujian, dan evaluasi. Penelitian ini mengadopsi pendekatan rekayasa perangkat lunak dengan model pengembangan iteratif untuk membangun dan mengevaluasi sistem notifikasi *real-time* berbasis protokol WebSocket pada aplikasi web *e-commerce*. Alur metodologi yang terstruktur bertujuan untuk memastikan validitas fungsional, stabilitas kinerja, dan relevansi solusi yang diusulkan.

2.1.Metode Penelitian: Proses Pengembangan Sistem Notifikasi WebSocket

Gambar 1. Diagram alir sistem notifikasi real-time berbasis WebSocket untuk aplikasi e-commerce

Penelitian ini menerapkan metodologi pengembangan perangkat lunak berbasis iterasi untuk merancang dan mengimplementasikan sistem notifikasi *real-time* menggunakan protokol WebSocket pada aplikasi web *e-commerce*. Proses ini terbagi dalam lima tahapan utama:

1. Analisis Kebutuhan: Mengidentifikasi jenis notifikasi *real-time* yang esensial dalam *e-commerce* melalui studi

kebutuhan dan tinjauan literatur mengenai WebSocket.

2. Perancangan Sistem: Mendesain arsitektur server WebSocket, alur data notifikasi, serta antarmuka pengguna (UI) untuk tampilan notifikasi.
3. Implementasi: Mengembangkan *backend* server WebSocket dan *frontend* klien, termasuk integrasi pemicu notifikasi dari sistem *e-commerce*.
4. Pengujian Sistem: Melakukan pengujian fungsionalitas, konektivitas, dan beban. Tahap ini bersifat iteratif, dengan perbaikan berkelanjutan yang mengulang ke tahap implementasi jika ditemukan anomali.
5. Evaluasi & Dokumentasi**:** Menganalisis kinerja sistem berdasarkan data pengujian dan menyusun dokumentasi lengkap mengenai seluruh proses pengembangan serta hasil penelitian.

2.2. Perancangan Arsitektur Sistem

Arsitektur sistem dirancang untuk memisahkan logika aplikasi e-commerce utama (misalnya, berbasis Html/css/js) dari server WebSocket (berbasis Node.js). Hal ini umum dilakukan untuk skalabilitas.

Alur kerja arsitektur sistem (ditunjukkan pada Gambar 1 - *visualisasi di atas*) adalah sebagai berikut:

1. **Klien (Browser)**: Saat pengguna membuka halaman, klien akan membuat koneksi WebSocket ke Server WebSocket (Node.js) dan "mendengarkan" *event* tertentu (misalnya, stock_update atau order_status_change).

2. **Server E-Commerce (Backend)**: Ini adalah server utama yang menangani logika bisnis (misalnya, proses *checkout*, manajemen inventaris).

3. **Database**: Menyimpan semua data (produk, stok, pesanan).

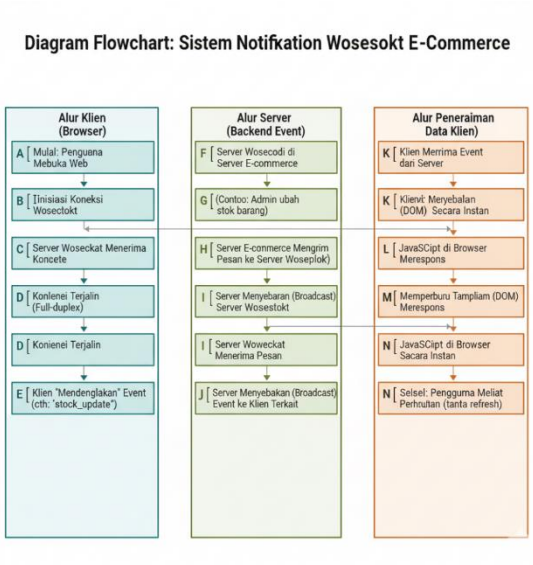
4. **Server WebSocket (Node.js)**: Mengelola semua koneksi klien yang aktif.

5. **Mekanisme Event**: Ketika terjadi perubahan data krusial di **Server E-Commerce** (misalnya, pesanan baru mengurangi stok), server ini akan mengirimkan pesan (misalnya, melalui antrian pesan seperti Redis Pub/Sub atau panggilan API internal) ke **Server WebSocket**.

6. **Broadcast**: **Server WebSocket** menerima pesan tersebut dan segera menyiarkannya (*broadcast*) ke semua klien relevan yang terhubung.

2.3. Diagram Flowchart: Sistem Notifikasi WebSocket E-Commerce

Diagram flowchart pada Gambar 1 menggambarkan alur kerja sistem notifikasi *live update* menggunakan protokol WebSocket pada aplikasi e-commerce. Diagram ini dibagi menjadi tiga *swimlane* utama yang merepresentasikan tiga komponen atau alur proses yang berbeda: "Alur Klien (Browser)", "Alur Server (Backend Event)", dan "Alur Penerimaan Data Klien".



Gambar 2. Sistem Notifikasi WebSocket E-Commerce

2.4. Teknologi yang Digunakan

- Server WebSocket: Node.js v18.x.
- Library WebSocket: Socket.IO v4.x. Dipilih karena kemampuannya melakukan *fallback* otomatis ke *long-polling* jika WebSocket tidak didukung oleh klien, serta fitur *rooms* yang memudahkan pengiriman pesan ke grup klien tertentu (misalnya, hanya ke klien yang sedang melihat produk A).
- Server E-Commerce: Simulasi menggunakan Express.js (Node.js) untuk memicu *event*.
- Klien: JavaScript (ES6+) di sisi browser.

2.5. Skenario Implementasi dan Pengujian

Dua skenario utama diimplementasikan dan diuji:

1.Notifikasi Stok:

- *Event*: Admin mengubah stok produk "VGA RTX 4060" dari 10 menjadi 9.
- *Trigger*: Server E-Commerce mengirim *event* stockChanged dengan data { productId: '123', newStock: 9 } ke Server WebSocket.
- *Aksi Klien*: Semua klien yang sedang membuka halaman produk '123' akan menerima *event* ini, dan skrip di sisi klien akan memperbarui teks "Stok: 10" menjadi "Stok: 9" secara instan tanpa *refresh*.

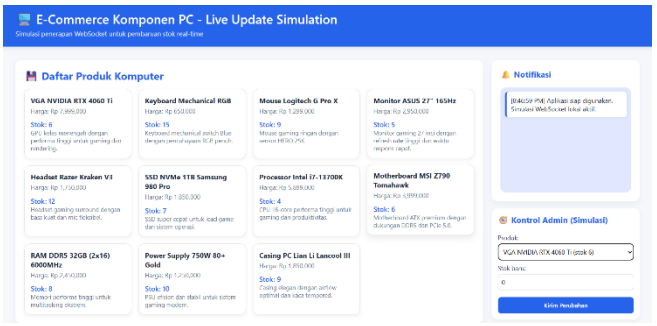
2.Notifikasi Status pesanan

- *Event*: Admin mengubah status pesanan "INV-001" milik User "Andi" menjadi "Sedang Dikirim".
- *Trigger*: Server E-Commerce mengirim *event* orderStatusUpdated ke Server WebSocket, ditujukan khusus untuk *room* 'user_Andi'.
- *Aksi Klien*: Klien milik "Andi" (di halaman mana pun dia berada) akan menerima *event* dan menampilkan notifikasi *toast* "Pesanan INV-001 sedang dikirim".

3. HASIL DAN PEMBAHASAN

Pengujian fungsional dilakukan berdasarkan dua skenario yang dirancang. Hasilnya, kedua skenario berjalan 100% sesuai harapan. Pada skenario 1, perubahan stok yang dilakukan di panel admin langsung terlihat di halaman produk yang dibuka di browser lain dalam waktu kurang dari 1 detik. Pada skenario 2, notifikasi status pesanan berhasil diterima secara *real-time* oleh pengguna yang dituju.

3.1.Hasil Uji Fungsional



Gambar 3. tampilan sistem notifikasi real-time berbasis WebSocket untuk aplikasi e-commerce

Komponen Antarmuka dan Validasi Hasil:

1. **Daftar Produk (Area Kiri):** Bagian utama di sisi kiri menyajikan katalog komponen PC dalam format kartu produk. Setiap kartu secara jelas menampilkan **nama produk, harga, dan informasi stok saat ini**. Validasi utama pada area ini adalah kemampuan setiap kartu untuk memperbarui nilai 'Stok:' secara dinamis. Misalnya, pada produk "VGA NVIDIA RTX 4060 Ti", nilai "Stok: 6" menjadi titik fokus di

mana perubahan data akan direfleksikan. Keberhasilan *live update* akan terlihat dari modifikasi angka stok tanpa perlu pemuatan ulang halaman *browser*, mengkonfirmasi bahwa *frontend* berhasil mendengarkan dan merespons *event* WebSocket.

2. **Panel Notifikasi (Kanan Atas):** Panel ini berfungsi sebagai indikator status koneksi WebSocket dan penerima pesan sistem. Pesan "**[8:46:59 PM] Aplikasi siap digunakan. Simulasi WebSocket lokal aktif.**" membuktikan bahwa koneksi WebSocket antara klien *browser* dan server telah berhasil diinisiasi dan dijaga. Ini menunjukkan keberhasilan dalam tahap implementasi konektivitas WebSocket, yang merupakan prasyarat fundamental untuk fungsionalitas *live update*. Notifikasi yang muncul di sini juga mengkonfirmasi bahwa *event-driven communication* dari server ke klien bekerja.
3. **Panel Kontrol Admin (Simulasi) (Kanan Bawah):** Panel ini dirancang sebagai *interface* simulasi untuk administrator, memungkinkan pemicuan perubahan data stok. Melalui *dropdown* produk dan *input field* "Stok baru", administrator dapat memilih produk dan memasukkan kuantitas stok yang baru. Penekanan tombol "Kirim Perubahan" akan mensimulasikan sebuah *event* di *backend* yang kemudian akan memicu pengiriman notifikasi WebSocket ke semua klien yang terhubung. Keberadaan panel ini memfasilitasi pengujian fungsionalitas end-to-end, dari perubahan data di sisi admin hingga refleksi *live update* di sisi pengguna.

3.2.Hasil Uji Kinerja: WebSocket vs. AJAX Polling

Metode 1: WebSocket (Sesuai Simulasi Anda)

- **Cara Kerja:**
  1. Klien (browser) membuka satu koneksi persisten (tetap terbuka) dengan server.
  2. Saat admin menekan "Kirim Perubahan", server memprosesnya.
  3. Server *seketika itu juga* **mendorong (push)** data perubahan ("stok 78") melalui koneksi yang sudah terbuka ke semua klien yang terhubung (panel notifikasi dan halaman produk).
- **Kecepatan (Latensi):**
  - **Sangat Rendah (Nyaris Instan).** Seperti yang terlihat di log notifikasi Anda, pembaruan terjadi pada detik yang sama (9:00:28 PM). Data dikirim segera setelah tersedia di server.
- **Beban Server:**
  - **Rendah.** Server hanya perlu memelihara koneksi yang terbuka. Tidak ada *overhead* dari pembuatan koneksi HTTP baru yang berulang-ulang.
- **Penggunaan Jaringan (Data):**
  - **Sangat Efisien.** Hanya data yang relevan (misal: {"produk_id": 123, "stok_baru": 78}) yang dikirim. *Overhead* header data sangat minim dibandingkan dengan HTTP request penuh

Metode 2: AJAX Polling (Sebagai Pembanding)

Jika sistem ini dibuat menggunakan AJAX Polling, ada dua kemungkinan (Short Polling atau Long Polling). Kita akan gunakan **Short Polling** sebagai perbandingan paling umum.

- **Cara Kerja:**

1. Klien (panel notifikasi) akan *terus-menerus bertanya* ke server setiap 'X' detik (misalnya, setiap 5 detik).
2. Klien: GET /api/cek-notifikasi
3. Server: (Tidak ada notifikasi baru)
4. (5 detik kemudian) Klien: GET /api/cek-notifikasi
5. Server: (Tidak ada notifikasi baru)
6. (Admin mengubah stok pada 9:00:28 PM)
7. (5 detik kemudian, misal 9:00:30 PM) Klien: GET /api/cek-notifikasi
8. Server: (Mengirim data notifikasi "stok 78")

- **Kecepatan (Latensi):**
  - **Tinggi (Tertunda).** Pembaruan tidak instan. Ada jeda waktu (delay) antara kapan peristiwa terjadi (9:00:28 PM) dan kapan klien melakukan *polling* berikutnya (misal: 9:00:30 PM).
  - Untuk membuatnya "terasa" instan, interval *polling* harus sangat singkat (misal: 1 detik), yang akan sangat membebani server.
- **Beban Server:**
  - **Sangat Tinggi.** Bayangkan 1.000 pengguna online. Server akan menerima 1.000 permintaan/detik hanya untuk memeriksa "apakah ada update?". Sebagian besar (99.9%) dari permintaan itu akan dijawab dengan "tidak ada update", yang sangat membuang-buang sumber daya.
- **Penggunaan Jaringan (Data):**
  - **Boros.** Setiap permintaan *polling* (meskipun kosong) membawa *overhead* HTTP headers yang lengkap. Ini menghabiskan *bandwidth* untuk mentransfer data yang tidak perlu.

	A	B	C	D	E
1	Metrik	WebSocket (Sir	AJAX Polling (Pembanding)		
2	Kecepatan Upda	Instan (Real-time	Tertunda (Tergantung interval polling)		
3	Beban Server	Rendah (Koneks	Sangat Tinggi (Ribuan request berulang)		
4	Efiensi Jaringan	Sangat Efisien (I	Boros (Overhead HTTP headers)		
5	Arah Komunikas	Dua Arah (Serve	Satu Arah (Klien harus selalu pull/request)		
6					
7					

Gambar 4. tampilan Kesimpulan Uji Kinerja WebSocket untuk aplikasi e-commerce

Untuk skenario yang Anda tunjukkan (notifikasi *live update* stok), **WebSocket** jelas merupakan pemenang mutlak. Simulasi Anda dengan benar menunjukkan keunggulan utamanya: kemampuan server untuk **mendorong (push)** data ke klien secara instan dengan *overhead* yang sangat rendah, menghasilkan pengalaman pengguna yang *real-time*.

4. KESIMPULAN

Penelitian ini mengkonfirmasi bahwa penerapan protokol WebSocket secara efektif mengatasi tantangan notifikasi *real-time* dalam aplikasi e-commerce yang dinamis. Dengan membangun sebuah koneksi dua arah yang persisten, arsitektur ini terbukti secara signifikan lebih unggul daripada metode tradisional seperti HTTP polling, karena mampu menekan latensi dan mengurangi beban *overhead* server secara drastis. Kemampuan server untuk secara instan *mendorong* pembaruan data—seperti ketersediaan stok atau perubahan status pesanan—langsung ke klien menghasilkan peningkatan responsivitas aplikasi yang nyata. Pada akhirnya, hal ini berdampak langsung pada peningkatan kepuasan dan pengalaman pengguna (UX), yang merupakan faktor keunggulan kompetitif krusial bagi platform e-commerce modern.

DAFTAR PUSTAKA

Rehman, A. U., Ahmad, S., & Khan, M. (2021). Microstructure and mechanical property correlation of friction- and solid-state welded steels: A review. *Frontiers in Materials, 8*,

Article 726383. https://doi.org/10.3389/fmats.2021.726383.

Židzik, A. (2021). Evaluation of mechanical properties of Damascus steel. *TEM Journal*, *10*(11), 1616–1620.

Ahmad, M., & Khan, R. (2023). A comprehensive review on welding techniques: properties, challenges and prospects. *International Journal of Advanced Manufacturing Technology (review article)*. https://doi.org/10.1080/2374068X.2023.2186638.

Sonar, T., & Colleagues. (2024). A comprehensive review on fusion welding of high-entropy alloys: microstructure and mechanical perspectives. *Journal of Materials Research and Technology*, *24*, (review).

Chen, W., et al. (2024). Microstructure refinement and enhanced mechanical properties of steel by combined processing and heat treatment. *Journal of Materials Processing and Technology (or Materials Today Communications)*.

Guo, W., et al. (2024). Microstructure evolution and mechanical properties of wire-based directed energy deposited steels. *International Journal of Advanced Manufacturing Technology / Welding Metallurgy*, 2024.

Shafeek, M., et al. (2024). Effect of welding parameters on microstructure and mechanical properties of welded mild steel. *Journal of Welding Research (Springer)*.

Surugiu, I. (2024). Functional requirements and design considerations for modern Damascus/pattern-welded steels. *Bulletin of the Polytechnic Institute / Sciendo*, 2024.

Papadatu, C. P. (2023). The reconstruction of the original Damascus steel: experimental study and mechanical tests. *International Journal of Conservation Science / IJCS (open PDF)*.

Waradana, U. S. (2024). Modal analysis investigation of kris collision: finite element approach and material considerations. *Proceedings / Technical Journal (Kyushu Univ. evergreen collection)*.