OPTIMASI PERFORMA API APLIKASI E-COMMERCE MENGGUNAKAN REDIS CACHING

Yuzaky Prilyansyah[1]*, Susanto[2]
[1,2]Teknik Informatika, Universitas Semarang
yuzakypril@gmail.com[1], susanto@usm.ac.id[2]

Submitted October 6, 2025; Revised December 1, 2025; Accepted December 3, 2025

Abstrak

Performa sistem merupakan salah satu faktor penting dalam pengembangan aplikasi e-commerce yang melayani ribuan permintaan data setiap detiknya. Tingginya intensitas permintaan pengguna sering menyebabkan beban query berlebih pada basis data, yang berdampak pada meningkatnya waktu respon dan menurunnya pengalaman pengguna. Salah satu solusi efektif untuk mengatasi permasalahan tersebut adalah dengan menerapkan mekanisme *caching*. Penelitian ini bertujuan untuk mengetahui pengaruh penggunaan Redis sebagai *caching layer* terhadap peningkatan performa API berbasis *Express.js* pada aplikasi e-commerce. Metode yang digunakan adalah eksperimen kuantitatif dengan tiga skenario pengujian, yaitu API tanpa Redis, API dengan Redis pada kondisi *cold cache*, dan API dengan Redis pada kondisi *warm cache*. Pengujian dilakukan menggunakan *load testing tool* k6 untuk mensimulasikan beban pengguna hingga 100 *virtual users*, dengan metrik yang diamati meliputi *response time*, *throughput*, dan *error rate*. Hasil pengujian menunjukkan bahwa penerapan Redis mampu menurunkan rata-rata *response time* hingga 69% dan meningkatkan *throughput* tiga kali lipat dibandingkan kondisi tanpa Redis. Performa terbaik diperoleh pada kondisi *warm cache*, di mana Redis telah menyimpan data dalam memori dan mampu memberikan respon dengan waktu rata-rata 32,68 ms. Penelitian ini membuktikan bahwa Redis merupakan solusi efektif untuk mengoptimalkan performa API pada sistem e-commerce dengan pola akses data yang tinggi dan berulang.
Kata Kunci : Redis, *Caching*, K6, ExpressJS, *E-Commerce*

Abstract

System performance is one of the important factors in the development of e-commerce applications that serve thousands of data requests every second. High intensity of user requests often leads to an overload of queries on the database, which has an impact on increased response times and decreased user experience. One of the effective solutions to overcome this problem is to implement a caching mechanism. This study aims to determine the effect of the use of Redis as a caching layer on improving the performance of Express.js-based APIs in e-commerce applications. The method used was a quantitative experiment with three test scenarios, namely API without Redis, API with Redis in cold cache condition, and API with Redis in warm cache condition. The test was carried out using the k6 load testing tool to simulate the user load of up to 100 virtual users, with the observed metrics including response time, throughput, and error rate. The test results showed that the implementation of Redis was able to reduce the average response time by up to 69% and increase throughput three times compared to the condition without Redis. The best performance is obtained in the warm cache condition, where Redis has stored data in memory and is able to provide a response with an average time of 32.68 ms. This study proves that Redis is an effective solution to optimize API performance in e-commerce systems with high and repetitive data access patterns.
Keywords : Redis, Caching, K6, ExpressJS, E-Commerce

1. PENDAHULUAN

Perkembangan teknologi informasi yang begitu cepat telah mendorong berbagai sektor untuk menyesuaikan diri dengan tuntutan digital, termasuk dalam ranah *e-commerce* [1]. Industri *e-commerce* di Indonesia terus menunjukkan pertumbuhan yang pesat dalam beberapa tahun terakhir [2]. Berdasarkan data dari Kementerian Perdagangan Republik Indonesia (2023),

jumlah pengguna *e-commerce* di Indonesia juga mengalami tren kenaikan signifikan, dari tahun 2020 hingga mencapai 58,63 juta pengguna pada 2023, dan diperkirakan akan terus tumbuh hingga 99,1 juta pengguna pada 2029. Tingkat penetrasi *e-commerce* pun terus meningkat, mencapai 21,56% pada tahun 2023 dan diproyeksikan naik hingga 34,84% pada tahun 2029. Fakta-fakta ini menunjukkan bahwa *e-commerce* memiliki peran yang semakin dominan dalam perekonomian digital Indonesia, sehingga kebutuhan akan sistem yang cepat, efisien, dan andal menjadi semakin krusial untuk menunjang pertumbuhan sektor ini [3].

Meski demikian, sebagian besar aplikasi *e-commerce* menghadapi masalah berupa tingginya latensi dan beban *query* yang berlebihan pada basis data, yang dapat mengakibatkan waktu respon yang lambat dan menurunkan pengalaman pengguna [4]. Kinerja yang buruk dapat berdampak langsung terhadap loyalitas pelanggan, bahkan menurunkan tingkat konversi transaksi [5] [6]. Strategi optimasi performa yang diperlukan tidak hanya berfokus pada infrastruktur, tetapi juga pada efisiensi pemrosesan data [7]. Salah satu solusi yang dapat diterapkan untuk mengatasi permasalahan ini adalah *caching*, yakni penyimpanan sementara data pada media berkecepatan tinggi sehingga dapat diakses kembali tanpa perlu melakukan query ulang ke basis data utama [8].

Redis (Remote Dictionary Server) menjadi salah satu teknologi *caching* yang populer karena berbasis *in-memory data store*, memiliki kecepatan baca dan tulis yang tinggi, serta mendukung berbagai jenis struktur data [9]. Dibandingkan dengan teknologi *caching* lain seperti *Memcached*, *Redis* lebih fleksibel karena mendukung berbagai tipe data, sistem persisten, dan mampu mengelola beban transaksi yang tinggi [10]. Dengan memanfaatkan *Redis*, performa *API e-commerce* dapat ditingkatkan melalui pengurangan beban query pada basis data dan percepatan waktu respon [11].

Beberapa penelitian terdahulu menunjukkan bahwa *caching* mampu memberikan peningkatan performa sistem secara signifikan [8], [12]. Namun, sebagian besar penelitian masih berfokus pada skenario caching secara umum tanpa melihat secara detail perbandingan performa dalam kondisi *cache* kosong (*cold cache*) dan *cache* terisi (*warm cache*). Padahal, perbedaan kondisi ini dapat memengaruhi hasil pengujian performa secara substansial [13].

Untuk mengukur performa pada ketiga kondisi tersebut digunakan *k6* sebagai *load testing tool* yang mampu mensimulasikan permintaan pengguna secara bersamaan, dengan data dummy produk agar pengujian merepresentasikan skenario pencarian data produk pada API *e-commerce* [14]. Penelitian ini diharapkan dapat menjawab *research gap* sebelumnya dengan memberikan analisis komparatif yang lebih komprehensif mengenai peran Redis dalam optimasi performa API berbasis Express.js, khususnya pada proses pengambilan data produk [15].

Penelitian ini difokuskan pada perbandingan hasil pengujian API dengan Redis dan tanpa Redis, meliputi kondisi *cold cache* dan *warm cache*. Evaluasi performa dilakukan berdasarkan tiga metrik utama, yaitu *response time, throughput*, dan tingkat kegagalan permintaan (*error rate*). Pengujian ini berfokus pada aspek waktu respon dan efisiensi akses data dari sisi pengguna, tanpa melibatkan pengukuran langsung terhadap konsumsi sumber daya server seperti CPU atau memori. Bagian berikutnya akan membahas metode, hasil pengujian, dan analisis performa Redis pada API *e-commerce.*

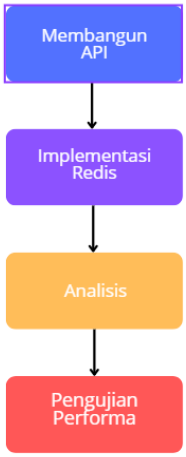
Oleh karena itu, penelitian ini bertujuan menganalisis pada penerapan Redis

terhadap peningkatan performa API e-commerce dalam tiga kondisi, yaitu API tanpa Redis, API dengan Redis pada kondisi *cold cache*, dan API dengan Redis pada kondisi *warm cache.*

Hasil dari penelitian ini diharapkan dapat memberikan manfaat baik secara teoretis maupun praktis. Secara teoretis, penelitian ini dapat memperkuat pemahaman mengenai efektivitas mekanisme *caching,* khususnya Redis, dalam meningkatkan kinerja API berbasis Node.js. Secara praktis, penelitian ini dapat menjadi acuan bagi pengembang aplikasi *e-commerce* dalam merancang arsitektur sistem yang efisien dan responsif, terutama pada layanan dengan intensitas permintaan data yang tinggi.

2. METODE PENELITIAN

Penelitian ini menggunakan pendekatan eksperimen kuantitatif, yang bertujuan untuk mengevaluasi pengaruh penerapan Redis sebagai *caching layer* terhadap performa API pada aplikasi *e-commerce* [12]. Eksperimen dilakukan melalui empat tahapan utama, yaitu:



Sumber : Dokumen Pribadi

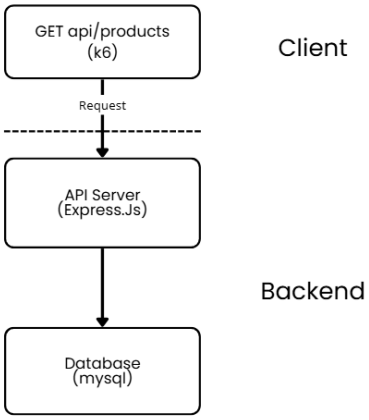
Gambar 1. Diagram Alur Penelitian

Perancangan Lingkungan Pengujian

Tahapan penelitian dimulai dengan pembuatan API katalog produk sebagai objek utama pengujian performa. API ini dikembangkan menggunakan *framework* Express.js yang berjalan di atas Node.js**,** dan terhubung dengan basis data relasional MySQL yang berisi 50 data *dummy* produk. Data dummy ini digunakan untuk mensimulasikan skenario pencarian produk pada sistem *e-commerce* yang lebih banyak melakukan pembacaan data daripada penulisan (*read-intensive).*

Seluruh komponen sistem dijalankan pada lingkungan lokal (localhost) dengan spesifikasi perangkat keras dan perangkat lunak yang dijaga agar tetap konstan selama proses eksperimen berlangsung, guna memastikan hasil pengujian dapat dibandingkan secara objektif.

Lingkungan pengujian dijalankan pada perangkat dengan spesifikasi prosesor *Intel Core i5-12450HX*, memori *RAM 12 GB*, dan sistem operasi *Windows 11 64-bit*. Backend API dikembangkan menggunakan *Node.js* versi *20.18.0*, dengan *MySQL 10.4.32* sebagai basis data utama. Proses pengujian performa dilakukan menggunakan *k6* sebagai *load testing tool* untuk mensimulasikan permintaan pengguna secara bersamaan.



Sumber : Dokumen Pribadi

Gambar 2. Arsitektur API Katalog Produk Tanpa Redis

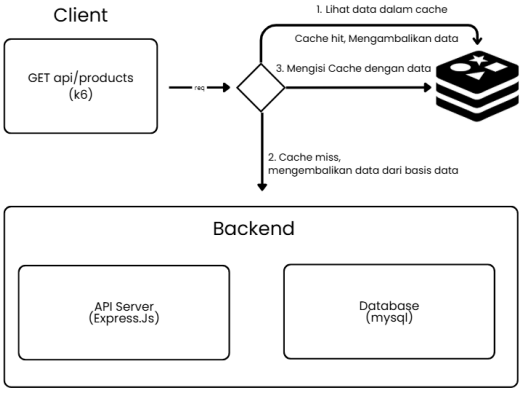
Arsitektur pada Gambar 2 menggambarkan alur permintaan data pada pengujian awal, di mana sistem belum menggunakan mekanisme *caching*. Klien (dalam hal ini *k6*

sebagai *load testing tool*) mengirimkan permintaan HTTP GET /api/*products* ke API Server yang dibangun menggunakan *Express.js*. Server kemudian memproses permintaan tersebut dengan mengambil data langsung dari basis data MySQL. Seluruh hasil query dikembalikan ke klien dalam format JSON. Arsitektur ini digunakan sebagai kondisi *baseline* untuk membandingkan performa sistem setelah Redis diterapkan.

Implementasi Redis

Pada tahap ini, Redis diterapkan sebagai *in-memory data store* yang berfungsi untuk menyimpan hasil *query* API agar dapat diakses kembali dengan lebih cepat tanpa perlu melakukan pengambilan ulang ke basis data utama. Implementasi *caching* dilakukan dengan menambahkan logika pemeriksaan cache pada endpoint GET /api/products. Saat permintaan diterima, sistem terlebih dahulu memeriksa apakah data sudah tersedia di Redis (*cache hit*). Jika data ditemukan, API langsung mengembalikan hasil tersebut ke klien tanpa mengakses basis data. Sebaliknya, apabila data tidak ditemukan (*cache miss*), API akan mengambil data dari basis data MySQL, mengembalikannya ke klien, serta menyimpannya ke dalam Redis dengan waktu kedaluwarsa (*Time To Live*, TTL) selama lima menit.

Gambar 3 menunjukkan alur kerja implementasi Redis sebagai *caching layer*, di mana Redis berperan sebagai lapisan perantara antara API *server* dan basis data untuk mempercepat proses pengambilan data.



Sumber : Dokumen Pribadi

Gambar 3. Implentasi Redis Sebagai Caching Layer

Pengujian Performa API

Tahap ketiga adalah pengujian performa API menggunakan k6. Pengujian dilakukan pada tiga kondisi sistem, yaitu API tanpa Redis, di mana permintaan data diproses langsung dari basis data; API Redis dengan cold cache, di mana Redis belum memiliki data saat pengujian dimulai sehingga permintaan pertama menghasilkan *cache miss*; dan API Redis dengan warm cache, di mana Redis telah berisi data hasil query sebelumnya sehingga sebagian besar permintaan menghasilkan *cache hit*.

Pengujian dilakukan menggunakan load testing tool k6, yang berfungsi untuk mensimulkasikan sejumlah permintaan pengguna secara bersamaan. Skenario beban yang digunakan terdisi atas tiga tahap, yaitu *ramp-up* (0-50 *virtual users* dalam 10 detik), *steady load* (naik ke 100 *virtual users* selama 50 detik), dan *ramp-down* (penurunan beban dalam 10 detik). Parameter yang diamati pada setiap pengujian meliputi *response time, throughput* (jumlah permintaan yang dapat diperoses per detik), serta *error rate* (tingkat kegagalan permintaan).

Seluruh pengujian dilakukan dalam kondisi lingkungan terkontrol, di mana tidak ada proses lain yang berjalan di latar belakang. Hal ini dilakukan untuk memastikan bahwa hasil pengujian sepenuhnya mencerminkan performa sistem yang diuji.

```
import http from 'k6/http';
import { check } from 'k6';
import { Rate } from 'k6/metrics';

// Custom metrics
const errorRate = new Rate('errors');

// Test configuration
export const options = {
  stages: [
    { duration: '10s', target: 50 },   // Ramp up to 50 VUs in 10s
    { duration: '50s', target: 100 },  // Stay at 100 VUs for 50s
    { duration: '10s', target: 0 },    // Ramp down to 0 VUs in 10s
  ],
  thresholds: {
    'http_req_duration': ['p(95)<200'], // 95th percentile should be < 200ms
    'http_req_failed': ['rate<0.05'],   // Error rate should be < 5%
    'errors': ['rate<0.05'],            // Custom error rate < 5%
  },
};

// Main test function
export default function () {
  // Make GET request to categories API
  const response = http.get('http://localhost:3000/products');
  // const response = http.get('http://localhost:3000/products-cache');

  // Record errors in custom metric
  errorRate.add(response.status !== 200);

  // Perform checks
  check(response, {
    'status is 200': (r) => r.status === 200,
    'response time < 200ms': (r) => r.timings.duration < 200,
    'response has body': (r) => r.body.length > 0,
    'content type is JSON': (r) =>
      r.headers['Content-Type'] &&
      r.headers['Content-Type']
      .includes('application/json'),
  });

  // Optional: Add small delay between requests to simulate real user behavior
  // sleep(1);
}

// Setup function (runs once at the beginning)
export function setup() {
  console.log('Starting performance test for products API');
  console.log('Target: http://localhost:3000/products');
  // console.log('Target: http://localhost:3000/products-cache');
  console.log('Scenario: 10s ramp-up to 50 VUs, 50s at 100 VUs, 10s ramp-down');
}

// Teardown function (runs once at the end)
export function teardown() {
  console.log('Performance test completed');
}
```

Sumber : Dokumen Pribadi

Gambar 4. Script K6 Untuk Pengujian Performa API

Gambar 4 di atas merupakan *Script k6* yang mendefinisikan tiga tahap pengujian beban (*stages*) untuk mensimulasikan lonjakan dan penurunan trafik secara realistis. Selain itu, disertakan pula metrik khusus seperti *error rate* serta *thresholds* untuk memastikan performa API tetap berada di bawah ambang batas 200 ms dengan tingkat kegagalan di bawah 5%.

Analisis Hasil Pengujian

Data hasil uji yang diperoleh dari k6 dianalisis secara deskriptif dengan menghitung rata-rata waktu response, nilai persentile ke-95, serta *throghput* dari setiap kondisi. Nilai *p95* digunakan untuk menggambarkan stabilitas sistem di bawah beban tinggi, sedangkan *error rate* menunjukkan tingkat keandalan API selama pengujian.

Hasil pengujian antar kondisi dibandingkan untuk menilai sejauh mana penerapan redis mampu meningkatkan performa API. Fokus analisis berada pada aspek kecepatan respon dan efisiensi akses data dari sisi pengguna (*client-side performance)*, tanpa mengukur konsumsi sumber daya *server* seperti CPU dan memori.

3. HASIL DAN PEMBAHASAN

Hasil Pengujian Perfoma API

Pengujian dilakukan untuk tiga kondisi sistem, yaitu API tanpa Redis, API dengan Redis pada kondisi *cold cache*, dan API dengan Redis pada kondisi *warm cache,* setiap pengujian dijalankan dengan scenario beban yang sama, yaitu peningkatan jumlah pengguna virtual hingga 100 Vus selama 70 detik waktu aktif. Metrik yang dikukur meliputi *response time, throughput*, serta *error rate.*

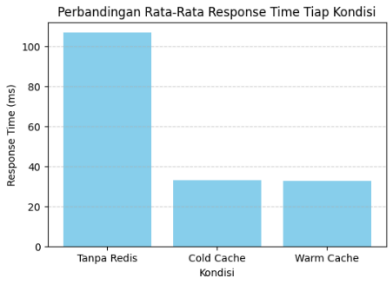
Tabel 1. Hasil Pengujian Menggunakan K6

Kondisi	Nilai	
Tanpa Redis	AVG	106.8 ms
	p95	157.8 ms
	Throughput	599 req/s
	Error Rate	0%
Cold Cache	AVG	32.85 ms
	p95	52.45 ms
	Throughput	1943 req/s
	Error Rate	0%
Warm Cache	AVG	32.68 ms
	p95	49.45 ms
	Throughput	1953 req.s
	Error Rate	0%

Sumber : Dokumen Pribadi

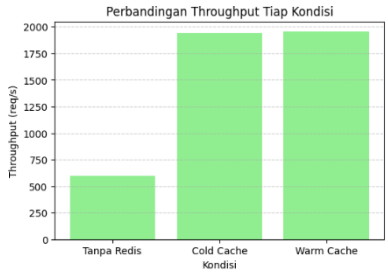
Berdasarkan Tabel 1, terlihat bahwa penggunaan Redis memberikan peningkatan performa yang sangat signifikan dibandingkan kondisi tanpa menggunakan redis. Pada kondisi tanpa redis, rata-rata response time mencapai 106,8 ms, sedangkan setelah diterapkan redis, rata-rata response time turun menjadi sekitar 32 ms pada kondisi *cold cache* dan 32,6 ms pada warm cache. Artinya, terjadi peningkatan kecepatan *response time* sekitar 69% lebih cepat dibandingkan tanpa menggunakan redis. Selain itu, throughput meningkat tiga kali lipat, dari 599 permintaan per detik menjadi lebih dari 1900 permintaan per detik. Nilai *error rate*

dari setiap kondisi adalah 0% ini menunjukan bahwa sistem bekerja dengan baik di setiap kondisi



Sumber : Dokumen Pribadi

Gambar 5. Grafik Rata Rata *Response Time* Tiap Kondisi



Sumber : Dokumen Pribadi

Gambar 6. Grafik *Troughput* Tiap Kondisi

Analisis Perbandingan Tiap Kondisi

1. API Tanpa Redis
   Kondisi tanpa Redis menunjukan performa paling rendah di antara ketiganya dengan rata-rata waktu respon 106,8 ms dan *throughput* 599 req/s . Hal ini disebabkan karena setiap permintaan data harus diambil langsung dari basis data utama tanpa proses penyimpanan sementara. Ketika jumlah pengguna meningkat, sistem memerlukan waktu tambahan untuk mengeksekusi *query* dan menngembalikan hasil, sehingga menambah latensi dan menurunkan *throughput.*
2. API Dengan Redis (*Cold Cache*)
   Pada kondisi *cold cache*, redis belum memiliki data yang tersimpan di awal pengujian, sehingga permintaan awal tetap diteruskan ke basis data. Namun setelah permintaan pertama, redis mulai menyimpan hasil *query* dalam memori, yang kemudian dimanfaatkan oleh permintaan berikutnya. Hal ini menjelaskan mengapa performa API *cold cache* sudah jauh lebih baik dibandingkan tanpa redis, karena efek *cache warm-up* terjadi cukup cepat di awal permintaan. Hasil ini juga memperlihatkan bahwa redis mampu menyesuaikan beban pengguna tinggi dengan mekanisme penyimpanan data yang efisien.
3. API Dengan Redis (*Warm Cache*)
   Kondisi warm cache menunjukan hasil terbaik dengan response time rata-rata 32,68 ms dan throughput 1953 req/s. karena data sudah tersimpan di memori redis sebelum pengujian dimulai, seluruh permintaan pengguna dapat langsung direspon tanpa *query* ke basis data. Perbedaan performa dengan *cold cache* relatif kecil, menandakan redis bekerja secara stabil begitu *cache* telah terisi penuh. Ini juga menunjukan bahwa manfaat utama redis terletak pada kecepatan akses data yang disimpan dalam memori.

Pembahasan

Hasil penelitian ini membuktikan bahwa penerapan redis secara signifikan mampu meningkatan performa API *e-commerce.* Redis mengurangi ketergantungan API terhadap *query* basis data, sehingga waktu akses data menjadi lebih cepat dan beban *server database* berkurang

Penurunan rata-rata *response time* hingga 69% dan peningkatan throughput tiga kali lipat memperlihatkan bahwa caching memiliki dampak nyata terhadap efisiensi sistem. Selain itu, hasil antara *cold cache* dan *warm cache* yang tidak terlalu berbeda menunjukan bahwa redis mampu mencapai kondisi optimal dengan cepat bahkan ketika *cache* belum sepenuhnya terisi. Hal ini mengindikasi efisiensi redis dalam membangun *cache hit ratio* yang tinggi dalam waktu singkat, terutama pada API dengan pola akses daya yang sering berulang seperti katalog produk.

Namun demikian, penelitian ini hanya berfokus pada metrik performa dari sisi pengguna. Pengukuran sumber daya *server* seperti CPU dan memori belum dilakukan, sehingga efisiensi sistem dari sisi infrastruktur belum dapat disimpulkan secara menyeluruh. Meski demikian, hasil pengujian cukup untuk membuktikan bahwa redis memberikan peningkatan signifikan terhadap kinerja API pada aplikasi *e-commerce* khususnya fitur katalog produk.

4. SIMPULAN

Penelitian ini menunjukkan bahwa penerapan *Redis caching* berpengaruh signifikan terhadap peningkatan performa API pada aplikasi *e-commerce*. Integrasi Redis terbukti mampu mempercepat waktu respon dan meningkatkan jumlah permintaan yang dapat dilayani sistem secara bersamaan (*throughput*), sehingga kinerja API menjadi lebih efisien dan responsif.

Dari hasil pengujian, penggunaan Redis baik pada kondisi *cold cache* maupun *warm cache* memberikan peningkatan performa yang jauh lebih baik dibandingkan dengan kondisi tanpa Redis. Hal ini mengindikasikan bahwa Redis dapat secara efektif mengurangi ketergantungan sistem terhadap query basis data, sekaligus mempertahankan kestabilan performa bahkan sejak awal penggunaan.

Dengan demikian, Redis dapat direkomendasikan sebagai solusi *caching* yang efektif untuk aplikasi berbasis Express.js dengan aktivitas baca data yang tinggi seperti *e-commerce*. Penelitian selanjutnya diharapkan dapat mengkaji aspek efisiensi sumber daya *server*, seperti penggunaan CPU dan memori, serta melakukan uji di lingkungan *cloud* atau integrasi dengan sistem *e-commerce* nyata, untuk memberikan pemahaman yang lebih menyeluruh mengenai dampak implementasi *caching* terhadap kinerja sistem secara keseluruhan.

DAFTAR PUSTAKA

[1] I. Gudiato, C., Sediyono, E., & Sembiring, “Analisis Sistem E - Commerce pada Shopee untuk meningkatkan daya saing,” *JIFOTECH (JOURNAL Inf. Technol.*, vol. 2, no. 1, pp. 6–10, 2022.

[2] R. Fandiyanto *et al.*, “Perkembangan E-Commerce Dari Masa Ke Masa: Sejarah, Tren, Dan Faktor-Faktor Yang Memengaruhi Kemajuan Transaksi Online Di Indonesia,” *J. Mhs. Entrep.*, vol. 4, no. 3, p. 448, Jun. 2025.

[3] R. Maharani and A. L. Prakoso, “Perlindungan Data Pribadi Konsumen Oleh Penyelenggara Sistem Elektronik Dalam Transaksi Digital,” *J. USM LAW Rev.*, vol. 7, no. 1, pp. 333–347, Mar. 2024.

[4] M. Rama Putra, M. A. Hasan, V. Stefanus, A. Bhaskoro Maghribi, M. Irawan, and R. Menzona, “Pengaruh Indexed Views Terhadap Performa Query Pada Basis Data Sql Server Dengan Data Skala Besar,” *JATI (Jurnal Mhs. Tek. Inform.*, vol. 9, no. 2, pp. 2586–2591, 2025.

[5] F. A. Laksono, S. Hadi Wijoyo, and A. R. Perdanakusuma, “Pengaruh Kualitas Layanan Terhadap Kepuasan Pelanggan dan Loyalitas Pengguna MyTelkomsel Dengan Menggunakan Model E-Service Quality dan E-Recovery Service Quality (Studi Kasus : Pengguna Aplikasi MyTelkomsel Malang),” *J. Pengemb. Teknol. Inf. dan ilmu Komput.*, vol. 4, no. 2, pp. 541–549, 2021.

[6] M. S. Ferreira, J. Antão, R. Pereira, I. S. Bianchi, N. Tovma, and N. Shurenov, “Improving real estate CRM user experience and

satisfaction: A user-centered design approach," *J. Open Innov. Technol. Mark. Complex.*, vol. 9, no. 2, p. 100076, Jun. 2023.

[7] D. M. Putri, W. Murtini, and P. Ninghardjanti, "Pengaruh persepsi promosi dan citra perusahaan terhadap loyalitas pelanggan aplikasi shopee," *JIKAP (Jurnal Inf. dan Komun. Adm. Perkantoran)*, vol. 8, no. 3, p. 238, May 2024.

[8] W. Jazuli, R. H. Ramadhan, R. Kharisma, and A. Rasyid, "Optimasi Kinerja Database Menggunakan Teknik Indexing dan Caching," *Gateaway*, vol. 1, no. 2, pp. 61–65, 2025.

[9] J. Kalajdjieski, M. Raikwar, N. Arsov, G. Velinov, and D. Gligoroski, "Databases fit for blockchain technology: A complete overview," *Blockchain Res. Appl.*, vol. 4, no. 1, p. 100116, Mar. 2023.

[10] S. Kanthed, "Redis vs. Memcached in Microservices Architectures: Caching Strategies," *Int. J. Multidiscip. Res. Growth Eval.*, vol. 4, no. 3, pp. 1084–1091, 2023.

[11] M. I. Zulfa, A. Fadli, and A. W. Wardhana, "Application caching strategy based on in-memory using Redis server to accelerate relational data access," *J. Teknol. dan Sist. Komput.*, vol. 8, no. 2, pp. 157–163, Apr. 2020.

[12] I. Nur Ramadhan and G. Saraswati, "Penerapan Database Redis Sebagai Optimalisasi Pemrosesan Kueri Data Pengguna Aplikasi SIRESMA Berbasis Laravel," *Technomedia J.*, vol. 8, no. 3, pp. 64–77, 2023.

[13] A. Papaioannou and K. Magoutis, "Addressing the Read-Performance Impact of Reconfigurations in Replicated Key-Value Stores," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 9, pp. 2106–2119, Sep. 2022.

[14] M. A. Oktafianto, B. T. Hanggara, and M. A. Akbar, "Analisis Perbandingan Performa Framework Web Server Nest JS dan Hapi JS Berbasis REST API," *J. Pengemb. Teknol. Inf. dan Ilmu Komput.*, vol. 9, no. 2, pp. 2548–964, 2025.

[15] M. V. Privalov and M. V. Stupina, "Improving web-oriented information systems efficiency using Redis caching mechanisms," *Indones. J. Electr. Eng. Comput. Sci.*, vol. 33, no. 3, pp. 1667–1675, Mar. 2024.