Jurnal Teknik dan Science
JTS Vol 5 No. 2 June 2026 | ISSN: 2828-7002 (Print), ISSN: 2828-6871 (online), Hal 146-156

Analisis Eksperimental Pengaruh Strategi Caching Redis terhadap Kinerja REST API pada Framework Laravel dan NestJS

Junaedi[1], Yan Risa Aspi Sururi[2]
Universitas Gunadarma

Article History
Received : 2026-03-07
Revised : 2026-04-23
Accepted : 2026-05-29
Published : 2026-07-30

Corresponding author*:
joned@staff.gunadarma.ac.id

Cite This Article: Junaedi, J., & Yan Risa Aspi Sururi. (2026). Analisis Eksperimental Pengaruh Strategi Caching Redis terhadap Kinerja REST API pada Framework Laravel dan NestJS. Jurnal Teknik Dan Science, 5(2), 146–156.

DOI:
https://doi.org/10.56127/jts.v5i2.2542

Abstract: The development of information technology has encouraged more services to interconnect through REST APIs. Under high read load, the same requests often access the relational database repeatedly, increasing latency and reducing throughput. This study aims to analyze the effect of Redis caching with a cache-aside pattern on REST API performance in the Laravel 12 and NestJS 11 frameworks. The method used is a quantitative experiment with a 2×2 factorial design combining framework and cache condition (no cache and Redis). Load testing was conducted using k6 at 50, 100, and 200 virtual users. Both APIs used the same MySQL dataset of 1,000 products, the same endpoints, and a 60-second cache TTL. The observed metrics include RPS, average latency, median, P95, error rate, and cache hit ratio. The results show that Redis provides a substantial performance improvement on NestJS. At 200 virtual users, average latency decreased by 89.8% and RPS increased by 111.3%, with a hit ratio of 99.7–99.9%. On Laravel, the hit ratio reached 100%, but latency improvement was only 6.7–10.8% at 50–100 virtual users and remained unstable at 200 virtual users. It can be concluded that Redis is beneficial for APIs with many read operations, but the magnitude of its benefit depends on the application runtime's ability to handle concurrency.

Keywords: Redis; REST API; Laravel; NestJS; performance evaluation.

PENDAHULUAN

Perkembangan teknologi informasi membuat perubahan perilaku dalam pencarian maupun pertukaran data antar sistem. Perubahan ini berdampak pada layanan yang menyediakan informasi kepada publik maupun antar aplikasi, khususnya melalui REST API. REST API banyak digunakan karena sederhana, berbasis HTTP, dan mudah diintegrasikan ke klien web maupun mobile (Fielding, 2000). Seiring bertambahnya jumlah pengguna yang mengakses layanan secara bersamaan, tantangan utamanya bukan lagi pada kelengkapan fitur semata, melainkan pada kemampuan sistem menjaga waktu respons yang masih dapat diterima.

Pada beban read-heavy, pola akses data biasanya timpang. Sebagian kecil endpoint dipanggil sangat sering, sementara isi datanya tidak selalu berubah setiap detik. Jika setiap permintaan tetap jatuh ke basis data relasional, biaya I/O, parsing hasil kueri, dan serialisasi respons menumpuk. Akibatnya latensi memanjang, throughput turun, dan pada titik tertentu mulai muncul timeout atau error (Jain, 1991). Kondisi ini umum ditemui pada katalog produk, data master, atau konten yang dibaca jauh lebih sering daripada ditulis.
Salah satu pendekatan yang sering digunakan untuk menekan masalah tersebut adalah caching. Redis menempati posisi penting karena menyimpan data di memori, mendukung struktur key-value, dan cocok untuk pola cache-aside (Carlson, 2013; Redis Ltd., n.d.). Di

ekosistem aplikasi modern, Laravel dan NestJS sama-sama menyediakan jalur integrasi Redis yang relatif matang (Laravel LLC, n.d.; NestJS, n.d.). Keduanya juga banyak diajarkan di perguruan tinggi serta dipakai di industri, sehingga pertanyaan komparatifnya menjadi relevan secara praktis maupun akademis.

Meskipun demikian, keputusan teknis sering diambil berdasarkan benchmark komunitas atau pengalaman personal. Angka yang beredar di internet biasanya membandingkan kecepatan "mentah" framework, tanpa menyamakan endpoint, isi tabel, TTL cache, maupun prosedur load testing. Padahal tanpa kontrol variabel semacam itu, sulit membedakan apakah perbedaan kinerja berasal dari Redis, dari ORM, atau dari model proses server. Penelitian ini hadir untuk mengisi kebutuhan bukti empiris yang lebih tertib.

METODE PENELITIAN

Desain Penelitian

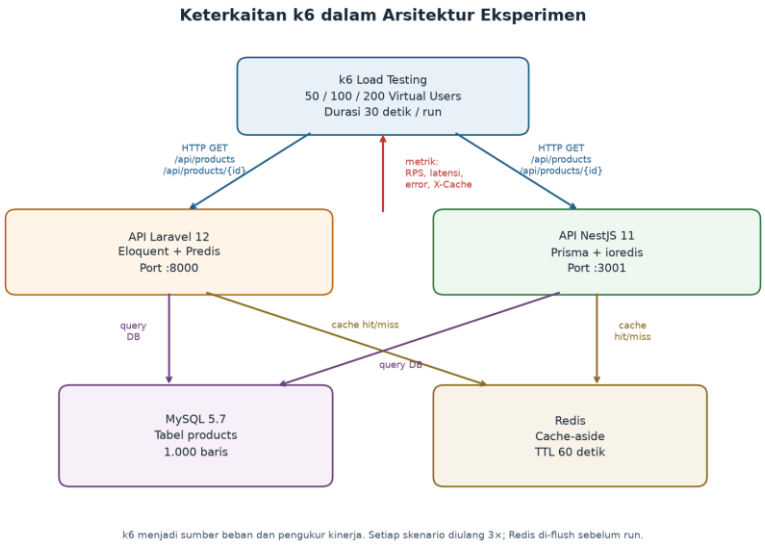
Penelitian memakai pendekatan kuantitatif eksperimental. Desainnya faktorial 2×2. Faktor A adalah framework aplikasi: Laravel 12 dengan Eloquent, serta NestJS 11 dengan Prisma. Faktor B adalah kondisi cache: tanpa cache dan Redis cache-aside. Untuk melihat perilaku pada beban berbeda, setiap kombinasi diuji pada 50, 100, dan 200 virtual users. Dengan demikian terdapat 12 kondisi pengukuran utama, masing-masing diulang tiga kali sehingga total pengujian terencana berjumlah 36.

Lingkungan Eksperimen

Eksperimen dijalankan pada lingkungan lokal agar variabel infrastruktur lebih mudah dikontrol. Rinciannya sebagai berikut.

Aspek	**Rincian**
Perangkat keras	Komputer lokal peneliti berbasis macOS
Sistem basis data	MySQL 5.7 (MAMP), host 127.0.0.1, port 8889
Sistem cache	Redis, host 127.0.0.1, port 6379
Framework A	Laravel 12, Eloquent, Predis
Framework B	NestJS 11, Prisma, ioredis
Alat load testing	k6
Orkestrasi	skrip shell lokal, tanpa Docker
TTL cache	60 detik

Kedua API dihubungkan ke database dan tabel yang sama, sehingga perbedaan hasil tidak disebabkan oleh perbedaan isi data maupun skema. Keterkaitan k6 dengan komponen eksperimen digambarkan pada Gambar 1.



Gambar 1. Keterkaitan k6 dengan API Laravel/NestJS, MySQL, dan Redis.

Pada Gambar 1 terlihat bahwa k6 berperan sebagai sumber beban sekaligus pengukur kinerja. k6 mengirim permintaan HTTP ke API Laravel atau NestJS sesuai skenario. Selanjutnya API memeriksa Redis terlebih dahulu; bila terjadi miss, data diambil dari MySQL lalu disimpan ke Redis. Respons API beserta header X-Cache dikembalikan ke k6 untuk dihitung menjadi RPS, latensi, error rate, dan perkiraan hit ratio.

Dataset / Objek Penelitian

Objek penelitian berupa REST API katalog produk. Dataset yang digunakan terdiri atas 1.000 baris pada tabel products, dengan atribut utama sku, name, description, price, stock, dan category. Endpoint yang diukur dipilih karena mewakili pola baca yang umum digunakan:

1. GET /api/products untuk mengambil 100 produk pertama yang diurutkan berdasarkan id;
2. GET /api/products/{id} untuk mengambil detail satu produk.

Alur cache-aside dibuat sederhana agar mudah disetarakan pada kedua framework. Pada saat permintaan masuk, aplikasi terlebih dahulu memeriksa Redis. Apabila data ditemukan (hit), respons langsung dikirimkan ke klien. Apabila data tidak ditemukan (miss), data diambil dari MySQL, kemudian disimpan ke Redis dengan TTL 60 detik, lalu dikirimkan ke klien. Pada operasi tulis, kunci yang terkait dihapus dari cache. Untuk skenario tanpa cache, mekanisme pembacaan dan penulisan Redis dimatikan melalui pengaturan konfigurasi.

Variabel Penelitian

Variabel Bebas

- Framework: Laravel dan NestJS
- Kondisi cache: tanpa cache dan Redis
- Tingkat konkurensi: 50, 100, dan 200 virtual users

Variabel Terikat

- Throughput dalam requests per second (RPS)
- Latensi rata-rata, median, dan P95 dalam milidetik
- Error rate
- Cache hit ratio pada skenario Redis

Variabel kontrol meliputi skema tabel, jumlah record, bentuk endpoint, TTL, skrip k6, serta mesin uji yang sama.

Skenario Pengujian

Empat kombinasi utama diuji secara sistematis:

1. Laravel — No Cache
2. Laravel — Redis
3. NestJS — No Cache
4. NestJS — Redis

Pada setiap kombinasi, beban dinaikkan bertahap dari 50, 100, hingga 200 virtual users. Durasi tiap pengujian adalah 30 detik. Setiap kondisi diulang tiga kali. Sebelum pengujian dimulai, isi Redis dikosongkan agar tidak terdapat sisa data dari percobaan sebelumnya.

Teknik Pengumpulan Data

Pengumpulan data dilakukan secara otomatis. k6 mengekspor ringkasan metrik ke berkas JSON. Dari berkas tersebut diambil nilai RPS, latensi, dan error rate. Selain itu, skrip membaca header X-Cache untuk menghitung perkiraan hit dan miss. Setelah seluruh pengulangan selesai, nilai pada setiap kondisi pengujian dirata-rata. Pendekatan ini dipilih agar proses pengukuran dapat diulang dengan prosedur yang sama.

Teknik Analisis Data

Analisis utama bersifat deskriptif. Hasil pengulangan disusun dalam bentuk tabel nilai rata-rata, kemudian dihitung peningkatan relatif Redis terhadap kondisi tanpa cache, serta ditampilkan melalui grafik. Perbaikan latensi dihitung sebagai persentase penurunan terhadap kondisi tanpa cache, sedangkan perbaikan RPS dihitung sebagai persentase kenaikan. Pembahasan tidak hanya didasarkan pada nilai rata-rata, tetapi juga memperhatikan P95 agar sebaran latensi pada ekor distribusi tidak terabaikan. Adapun uji statistik inferensial, seperti ANOVA atau Kruskal-Wallis, dapat digunakan pada penelitian lanjutan setelah jumlah pengulangan diperbesar.

Prosedur Replikasi Singkat

Agar penelitian mudah diulang, urutan kerja diringkas sebagai berikut. Pertama, basis data MySQL dan Redis dipastikan berjalan, lalu dataset produk diisi. Kedua, API Laravel atau NestJS dijalankan dengan flag CACHE_ENABLED sesuai skenario. Ketiga, Redis dikosongkan. Keempat, k6 dijalankan pada tingkat virtual users yang ditentukan selama 30 detik. Kelima, langkah ketiga dan keempat diulang hingga tiga kali. Keenam, berkas JSON hasil uji diagregasi menjadi nilai rata-rata per kondisi. Prosedur yang sama dipakai untuk seluruh kombinasi framework dan kondisi cache, sehingga perbedaan antar kondisi dapat dibandingkan secara lebih adil.

HASIL DAN PEMBAHASAN

Implementasi

Implementasi dilakukan dengan menyiapkan satu database MySQL dan satu instance Redis yang digunakan bersama oleh kedua API. Pada sisi Laravel, akses data dilakukan melalui Eloquent, sedangkan akses Redis menggunakan Predis. Pada sisi NestJS, akses data dilakukan melalui Prisma, sedangkan layanan cache menggunakan ioredis. Bentuk endpoint pada kedua API disamakan, sehingga skrip pengujian tidak perlu dibuat berbeda.

Untuk mendukung jalannya eksperimen, ditambahkan endpoint /api/health guna memeriksa koneksi database dan Redis, serta /api/experiment/flush-cache guna mengosongkan cache sebelum pengukuran. Dataset sebanyak 1.000 produk diisi satu kali, kemudian dibaca oleh kedua API. Dengan cara demikian, perbedaan hasil lebih diarahkan pada perbedaan framework, runtime, dan perlakuan cache, bukan pada perbedaan data.

Hasil Pengujian

Hasil lengkap rata-rata tiga pengulangan disajikan pada Tabel 2.

Tabel 2. Rata-rata kinerja per skenario (3 pengulangan)

Framework	Cache	VU	RPS	Avg (ms)	Med (ms)	P95 (ms)	Error	Hit ratio
Laravel	No Cache	50	125,09	348,08	321,13	530,84	0,0000	—

Framework	Cache	VU	RPS	Avg (ms)	Med (ms)	P95 (ms)	Error	Hit ratio
Laravel	No Cache	100	132,57	695,24	692,35	968,13	0,0000	—
Laravel	No Cache	200	146,69	848,35	876,94	1.151,58	0,0416	—
Laravel	Redis	50	138,72	310,39	275,65	549,80	0,0000	100%
Laravel	Redis	100	141,65	648,44	608,91	1.001,32	0,0000	100%
Laravel	Redis	200	137,67	894,05	852,39	1.605,17	0,0447	100%
NestJS	No Cache	50	702,47	20,44	8,49	47,08	0,0000	—
NestJS	No Cache	100	1.411,68	20,48	10,20	63,15	0,0000	—
NestJS	No Cache	200	1.640,82	71,57	49,33	139,44	0,0000	—
NestJS	Redis	50	916,30	4,09	2,27	10,71	0,0000	99,7%
NestJS	Redis	100	1.813,80	4,72	2,08	14,03	0,0000	99,8%
NestJS	Redis	200	3.466,66	7,30	2,52	27,68	0,0000	99,9%

Pada kondisi tanpa cache, NestJS sudah lebih cepat dibandingkan Laravel. Di 50 virtual users, latensi rata-rata NestJS hanya 20,44 ms, sedangkan Laravel berada di 348,08 ms. Ketika jumlah virtual users dinaikkan, keduanya mengalami tekanan. Namun kenaikan latensi pada Laravel terlihat lebih tajam. Perbedaan baseline ini perlu diperhatikan sebelum Redis diaktifkan, karena akan memengaruhi cara membaca manfaat Redis pada masing-masing framework.

Setelah Redis diaktifkan, NestJS mengalami peningkatan kinerja pada semua tingkat virtual users. Pada 200 virtual users, latensi rata-rata turun dari 71,57 ms menjadi 7,30 ms, sedangkan RPS naik dari 1.640,82 menjadi 3.466,66. Hit ratio berada pada kisaran 99,7–

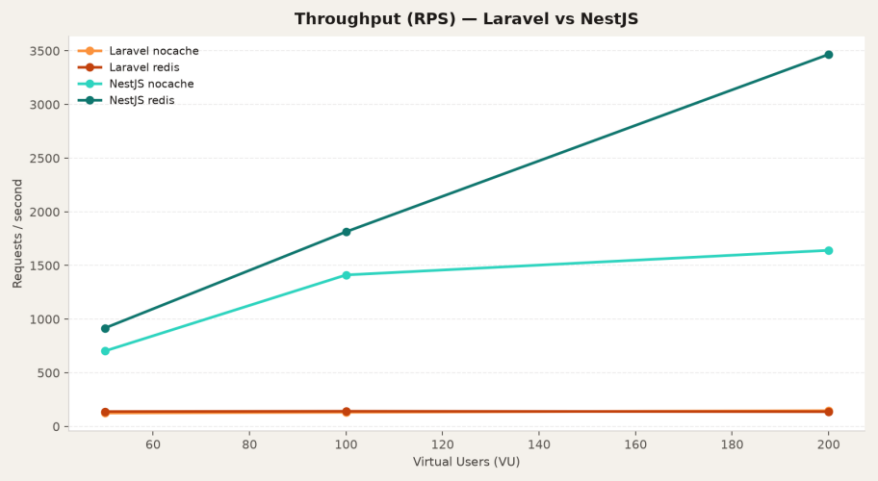
99,9%. Hasil tersebut menunjukkan bahwa sebagian besar permintaan telah dilayani melalui Redis.

Pada Laravel, pola yang muncul berbeda. Hit ratio mencapai 100%, sehingga Redis dapat dikatakan bekerja dengan baik setelah fase pengisian awal. Akan tetapi, perbaikan latensi rata-rata hanya 10,8% pada 50 virtual users dan 6,7% pada 100 virtual users. Pada 200 virtual users, latensi rata-rata dan P95 justru meningkat, disertai error rate sekitar 4,5%. Hal ini menunjukkan bahwa keberhasilan cache di tingkat data belum tentu menghasilkan peningkatan kinerja secara keseluruhan.

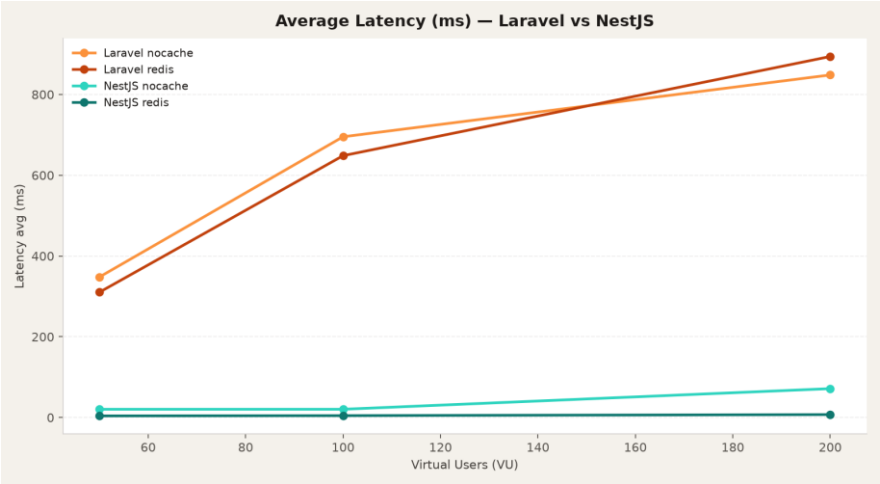
Tabel 3. Peningkatan relatif Redis terhadap kondisi tanpa cache

Frame work	VU	Penurunan Avg	Penurunan P95	Kenaikan RPS	Hit ratio
Laravel	50	10,8%	−3,6%	10,9%	100%
Laravel	100	6,7%	−3,4%	6,8%	100%
Laravel	200	−5,4%	−39,4%	−6,1%	100%
NestJS	50	80,0%	77,2%	30,4%	99,7%
NestJS	100	76,9%	77,8%	28,5%	99,8%
NestJS	200	89,8%	80,2%	111,3%	99,9%

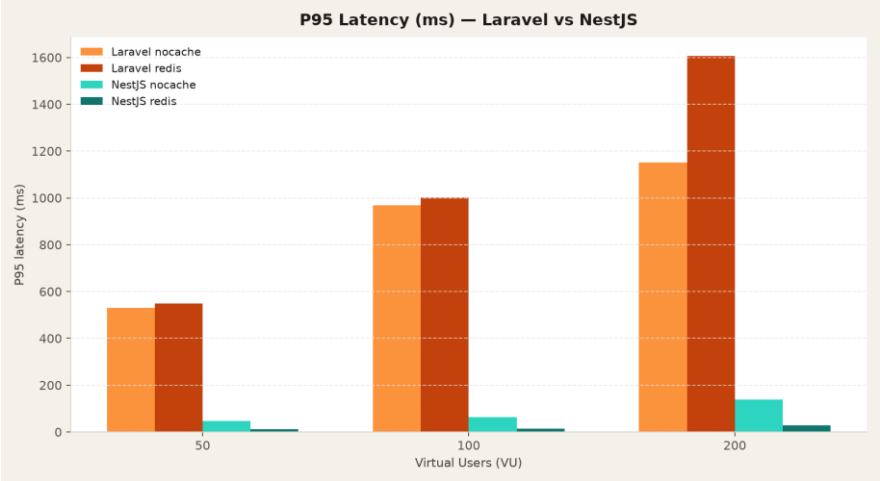
Gambar 2 sampai Gambar 5 menampilkan tren visual hasil pengujian.



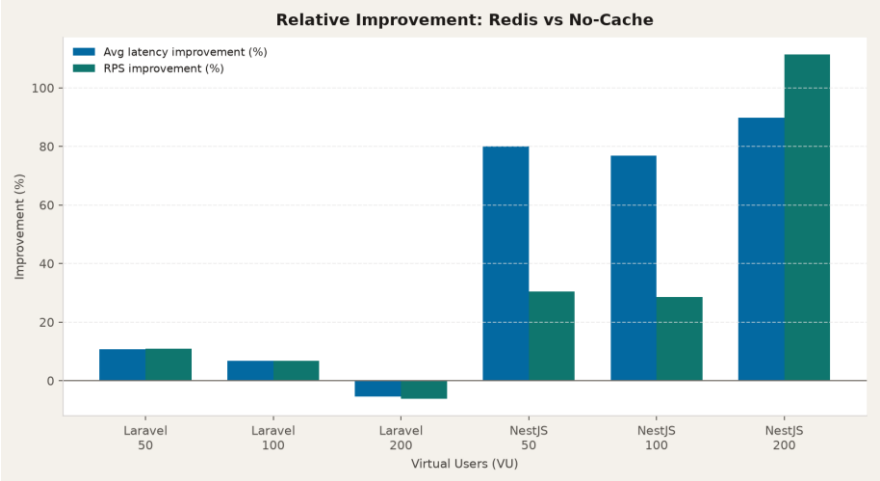
Gambar 2. Throughput (RPS) pada berbagai tingkat virtual users.



Gambar 3. Latensi rata-rata (ms) pada berbagai tingkat virtual users.



Gambar 4. Latensi P95 (ms) pada berbagai tingkat virtual users.



Gambar 5. Peningkatan relatif Redis terhadap kondisi tanpa cache.

Jika dilihat per tingkat beban, pola NestJS relatif stabil. Pada 50 virtual users, latensi rata-rata turun dari 20,44 ms menjadi 4,09 ms. Pada 100 virtual users, penurunan masih tajam, yaitu dari 20,48 ms menjadi 4,72 ms. Pada 200 virtual users, latensi tanpa cache naik menjadi 71,57 ms, kemudian turun kembali menjadi 7,30 ms setelah Redis digunakan. Dengan demikian, manfaat Redis tetap terlihat meskipun beban semakin tinggi.

Pada Laravel, manfaat Redis paling terlihat pada beban menengah ke bawah. Di 50 virtual users, RPS naik dari 125,09 menjadi 138,72 dan latensi rata-rata turun dari 348,08 ms menjadi 310,39 ms. Di 100 virtual users, perbaikan masih ada, tetapi semakin kecil. Di 200 virtual users, RPS tidak naik, P95 memanjang, dan mulai muncul error. Dari pola tersebut, batas kinerja diperkirakan sudah berpindah ke sisi aplikasi atau server, bukan hanya pada basis data.

Dari seluruh tabel dan grafik, terdapat dua temuan utama. Pertama, Redis memberi manfaat besar dan stabil pada NestJS. Kedua, pada Laravel dalam konfigurasi uji ini, cache berfungsi secara teknis tetapi belum cukup untuk menggeser bottleneck utama ketika konkurensi tinggi.

Pembahasan

Interpretasi Hasil

Hasil NestJS dapat dijelaskan secara cukup jelas. Ketika Redis aktif, sebagian besar permintaan dilayani dari memori. Karena hit ratio-nya tinggi, MySQL tidak lagi menjadi jalur utama. Runtime Node yang terbiasa menangani banyak koneksi dalam satu proses tampak mampu memanfaatkan latensi Redis yang rendah, sehingga RPS juga meningkat. Pada 200 virtual users, kenaikan RPS bahkan lebih dari dua kali lipat.

Hasil Laravel perlu dibaca lebih hati-hati. Apabila hanya dilihat dari hit ratio, implementasinya tampak berhasil. Namun dari sisi latensi dan RPS, peningkatannya masih terbatas. Perbaikan pada 50 dan 100 virtual users memang ada, tetapi tidak besar. Pada 200 virtual users, P95 justru memburuk. Dari hasil tersebut, antrian pada server uji Laravel (php artisan serve) diduga lebih berpengaruh daripada latensi kueri MySQL. Redis mampu mengurangi beban database, tetapi belum cukup untuk mengatasi keterbatasan model proses aplikasi pada beban tinggi.

Temuan ini juga mengingatkan bahwa rata-rata saja tidak cukup. Pada beberapa kondisi Laravel, pergerakan rata-rata dan P95 tidak selalu searah. Pengguna akhir biasanya merasakan ekor latensi lebih kuat daripada nilai tengah, sehingga P95 menjadi indikator yang perlu diperhatikan.

Perbandingan dengan Penelitian Sebelumnya

Secara umum, hasil NestJS sejalan dengan temuan Patel et al. (2025) dan Prasetyo et al. (2024) bahwa Redis bermanfaat untuk API read-heavy. Perbedaannya, penelitian ini membandingkan dua framework sekaligus dengan perlakuan Redis yang sama. Dari perbandingan tersebut terlihat bahwa manfaat Redis dapat berbeda tergantung runtime yang digunakan.

Berbeda dengan Swapna et al. (n.d.) yang menitikberatkan REST dan gRPC, penelitian ini lebih fokus pada Laravel dan NestJS. Berbeda pula dengan Okami101 (2026), penelitian ini tidak hanya menampilkan kecepatan absolut, tetapi juga peningkatan relatif

setelah Redis digunakan. Dengan demikian, pembahasan diarahkan pada kapan Redis sangat membantu dan kapan manfaatnya tampak terbatas, bukan sekadar daftar peringkat kecepatan.

Implikasi Praktis

Dari sisi praktis, Redis tetap layak digunakan pada endpoint yang banyak dibaca, terutama bila aplikasi berjalan pada runtime yang mampu menangani konkurensi dengan baik. Namun keberhasilan caching tidak cukup dinilai hanya dari hit ratio. Nilai RPS, P95, dan error rate juga perlu diperhatikan bersama.

Hasil pada Laravel tidak berarti Redis gagal. Pada konfigurasi server uji yang masih berbasis proses tunggal, manfaat Redis memang terbatas. Sebelum digunakan pada lingkungan produksi, pengujian Laravel sebaiknya diulang dengan PHP-FPM/Nginx atau Laravel Octane. Adapun pada NestJS, pola cache-aside Redis terlihat cukup efektif untuk endpoint yang bersifat read-heavy.

Implikasi Akademis

Dari sisi akademis, penelitian ini menegaskan pentingnya kontrol variabel pada studi komparatif framework. Banyak klaim kinerja di ruang publik cenderung menempatkan framework sebagai pemenang atau pecundang absolut. Padahal ketika perlakuan optimasi seperti Redis diuji secara setara, yang muncul justru interaksi antara strategi caching dan karakteristik runtime. Pendekatan peningkatan relatif memberi cara baca yang lebih adil untuk kajian Software Performance Engineering pada web API modern.

Keterbatasan Penelitian

Penelitian ini masih memiliki beberapa keterbatasan. Pertama, model proses server uji belum setara, karena Laravel dijalankan dengan php artisan serve, sedangkan NestJS memakai server HTTP Node. Karena itu perbandingan absolut lintas framework perlu dibaca hati-hati; penekanan analisis lebih pada peningkatan relatif di dalam masing-masing framework.

Kedua, dataset yang digunakan hanya 1.000 baris dan pengujian dilakukan di lingkungan lokal. Kondisi ini belum tentu sama dengan produksi yang memakai load balancer, banyak instance, atau basis data terpisah secara geografis. Ketiga, pemantauan CPU, memori, dan latensi kueri MySQL belum dilakukan secara mendalam, sehingga penjelasan bottleneck masih bersifat inferensi berbasis pola metrik klien.

Keempat, penelitian ini masih terbatas pada pola cache-aside dengan TTL 60 detik. Variasi TTL, write-through, write-back, maupun strategi invalidasi yang lebih kompleks belum diuji. Kelima, analisis masih bersifat deskriptif; uji beda statistik pada data pengulangan yang lebih banyak belum diterapkan.

KESIMPULAN

Berdasarkan hasil eksperimen, kesimpulan penelitian dapat disusun sesuai rumusan masalah.

Pertama, Redis cache-aside memberikan peningkatan kinerja yang cukup besar pada NestJS. Pada 200 virtual users, latensi rata-rata turun hingga 89,8% dan RPS naik hingga

111,3%, dengan hit ratio mendekati penuh. Pada Laravel, Redis tetap berfungsi dengan baik karena hit ratio mencapai 100%, namun pengaruhnya terhadap latensi dan RPS masih terbatas, bahkan belum stabil pada beban 200 virtual users.

Kedua, pada tingkat beban yang sama, peningkatan kinerja Redis pada NestJS berbeda dengan Laravel. NestJS memperoleh peningkatan yang jauh lebih besar.

Ketiga, Redis bermanfaat untuk API yang banyak melakukan operasi baca, tetapi besar kecilnya manfaat tersebut bergantung pada kemampuan runtime dalam menangani konkurensi. Oleh karena itu, penerapan caching sebaiknya dipertimbangkan bersama dengan pilihan server aplikasi yang digunakan. Hasil penelitian ini menunjukkan bahwa Redis memang bermanfaat, terutama pada NestJS. Namun demikian, penambahan Redis tidak serta-merta memberikan peningkatan kinerja yang sama pada setiap framework.

DAFTAR PUSTAKA

Carlson, J. L. (2013). Redis in Action. Manning Publications.

Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures (Doctoral dissertation). University of California, Irvine.

Grafana Labs. (n.d.). k6 documentation. https://grafana.com/docs/k6/

Jain, R. (1991). The art of computer systems performance analysis: Techniques for experimental design, measurement, simulation, and modeling. Wiley.

Laravel LLC. (n.d.). Cache. https://laravel.com/docs/cache

NestJS. (n.d.). Caching. https://docs.nestjs.com/techniques/caching

Okami101. (2026). A 2026 benchmark of main web API frameworks. https://www.okami101.io/blog/web-api-benchmarks-2026/

Patel, A., et al. (2025). Optimizing REST API response time in Spring Boot: An empirical evaluation of caching strategies. International Journal of Innovative Research in Technology. https://ijirt.org/publishedpaper/IJIRT194648_PAPER.pdf

Prasetyo, A., et al. (2024). Analisis kinerja cache Redis pada RESTful API berbasis Express.js. Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer. https://j-ptiik.ub.ac.id/index.php/j-ptiik/article/download/15978/7063

Redis Ltd. (n.d.). Cache-aside with Redis. https://redis.io/docs/latest/develop/use-cases/cache-aside/

Swapna, G., Punith Kumar, S. V., & Susmitha, E. (n.d.). Relix: A comparative latency framework for REST and gRPC using Redis acceleration. IRJAEH. https://irjaeh.com/index.php/journal/article/download/1718/1579