SMATIKA : STIKI Informatika Jurnal
Vol. 15, No. 2, Desember 2025, pp. 322~330
ISSN: 2087-0256, e-ISSN: 2580-6939

Analisis Komparasi Web Stack Berbasis JavaScript pada Fase Konstruksi Aplikasi: MEAN, MERN, dan MEVN

Koko Wahyu Prasetyo [1*]
Bima Reynaldi Sumitro [2]

[1,2]Fakultas Sains Teknologi, Jl. Raya Tidar No.100, Karangbesuki, Kec. Sukun, Kota Malang, Jawa Timur 65146, Universitas Bhinneka Nusantara, Indonesia
[1] koko@ubhinus.ac.id, [2] 181111078@mhs.stiki.ac.id

***Penulis Korespondensi:**
Koko Wahyu Prasetyo
koko@ubhinus.ac.id

Abstrak

Perkembangan pesat pada *framework* dan *stack* berbasis JavaScript memudahkan sinkronisasi logika *frontend* dan *backend*, tetapi menimbulkan dilema bagi entitas pengembang aplikasi kecil-menengah yang harus memilih *stack* teknologi berdasarkan keterbatasan sumber daya. Studi-studi terdahulu cenderung mengevaluasi Angular, React, atau Vue secara terpisah, berfokus pada metrik kinerja atau tampilan, dan mengabaikan dimensi kemudahan pemeliharaan. Akibatnya, penelitian empiris sebagai panduan keputusan pemilihan *stack* Javascript masih terbatas. Penelitian ini mengisi kesenjangan tersebut dengan membandingkan MEAN, MERN, dan MEVN melalui empat indikator kuantitatif: *lines of code* (LOC), skor Lighthouse, kedalaman call-stack, dan konsumsi memori. Eksperimen komparasi dilakukan terhadap tiga prototipe sistem informasi yang dikonstruksi secara identik. Hasil eksperimen menunjukkan MERN menghasilkan LOC dan konsumsi memori paling rendah, MEVN memiliki *call-stack* paling sederhana, sedangkan ketiganya menunjukkan skor kinerja nyaris identik. Temuan penelitian ini dikonfirmasikan dengan literatur terkini dan menunjukkan bahwa aspek pemilihan *framework* terkait modularitas, efisiensi memori, dan kompleksitas pemeliharaan lebih menentukan daripada aspek performa kinerja semata.
Kata Kunci: Angular; JavaScript; React; Tumpukan Web (Web Stack); Vue.

Abstract

The rapid evolution of JavaScript frameworks and stacks streamlines frontend and backend integration, yet forces small-to-medium teams to choose a technology stack that fits their resource constraints. Prior studies typically evaluate Angular, React, or Vue in isolation, focus on rendering-speed metrics, and overlook maintainability aspects, leaving limited empirical guidance for holistic stack selection. Addressing this gap, the present work compares MEAN, MERN, and MEVN on identically featured application prototypes using four quantitative indicators: lines of code (LOC), Lighthouse score, call-stack depth, and memory footprint. Experiments reveal that MERN achieves the lowest LOC and memory usage, MEVN exhibits the shallowest call stack, and all three stacks attain near-identical performance scores. By relating these findings to recent literature, this study demonstrates that trade-offs among modularity, memory efficiency, and maintenance complexity could outweigh performance differences.
Keywords: Angular; JavaScript; React; Vue; Web Stack.

1. Pendahuluan

Pada masa awal kegiatan pengembangan aplikasi web, umumnya terdapat satu atau beberapa pengembang aplikasi menangani seluruh proses pembuatan aplikasi tanpa pembagian tugas terstruktur. Saat ini, peran tersebut terdistribusi secara lebih spesifik: *frontend developer* merancang antarmuka pengguna, sedangkan *backend developer* memproses data yang dimasukkan pengguna dan menyimpannya ke basis data [1]. Segmentasi ini mendorong munculnya beragam teknologi dan bahasa pemrograman.

Untuk sisi belakang (*backend*), terdapat beragam bahasa pemrograman yang dapat dipilih dengan berbasis Python [2], JavaScript [3], GoLang [4][5] , Java [6], maupun sejumlah framework yang berbasis PHP [7][8][9]. Sedangkan pada sisi depan (*frontend*), pilihan didominasi oleh JavaScript dan turunan TypeScript [10]. Berdasarkan karakteristik teknis JavaScript yang dapat dipakai di kedua sisi aplikasi, sejumlah pengembang memilih satu bahasa seragam sepanjang siklus hidup aplikasi [11]. Konsep ini melahirkan tiga tech stack berbasis JavaScript: MEAN (MongoDB,

Express.js, Angular, Node.js) [12], MERN (MongoDB, Express.js, React, Node.js) [13], dan MEVN (MongoDB, Express.js, Vue, Node.js) [14].

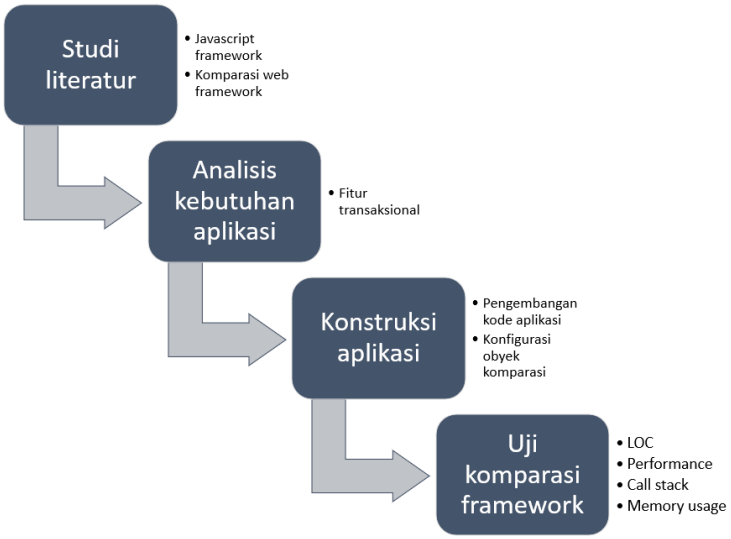
Ketiga stack tersebut berbagi tiga komponen utama: MongoDB sebagai basis data NoSQL yang mendukung JSON, Express.js sebagai web *framework* di atas Node.js, serta Node.js sebagai *runtime environment* JavaScript yang dibangun di atas mesin V8 Chrome [15]. Perbedaannya hanya terletak pada kerangka kerja *frontend*: Angular, React, atau Vue. Masing-masing menawarkan fitur unggulan berbeda, antara lain: jumlah baris kode, pola komponen, manajemen *state*, performa, *lazy loading*, penggunaan transpiler, dan fitur khas lainnya [16].

Studi komparatif sebelumnya berfokus pada evaluasi kerangka kerja Angular, React, dan Vue [17] tanpa menilai keseluruhan stack secara terpadu: mulai dari basis data, *runtime*, hingga pola arsitektur. Selain itu, metrik yang digunakan terbatas pada waktu render atau skor Lighthouse. Sedangkan dimensi *maintainability* seperti panjang kode, kedalaman *call stack*, dan efisiensi memori belum diteliti lebih lanjut. Komparasi ini membutuhkan indikator properti internal dan eksternal sebagai acuan kerangka kerja. Penelitian [18] dan [19] melakukan eksperimen menggunakan metrik seperti penggunaan memori, jumlah fungsi yang dipanggil per operasi CRUD, dan waktu eksekusi. Metrik konvensional seperti LOC (*lines of code*) juga masih relevan untuk digunakan dalam komparasi performa aplikasi [20]. Penelitian [21] melakukan komparasi performa antara aplikasi berbasis framework dan non-framework. Sedangkan [22] dan [23] mengevaluasi framework web melalui metrik baris kode, performa di peramban, serta struktur kode MVC dan OOP. Kondisi ini menciptakan kesenjangan pengetahuan dan terbatasnya referensi empiris mengenai pemilihan *framework*, terutama di lingkungan di mana pengembangan aplikasi kerap dipengaruhi oleh keterbatasan sumber daya pengembang dan infrastruktur.

Penelitian ini memberikan dua kontribusi utama untuk merespon kesenjangan tersebut. Penelitian ini menyajikan evaluasi empiris multidimensional terhadap MEAN, MERN, dan MEVN dengan menggunakan empat metrik tambahan pada fase konstruksi prototipe aplikasi yang identik: LOC, kinerja aplikasi, jumlah pemanggilan fungsi, dan konsumsi sumber daya memori. Penelitian ini juga merumuskan panduan praktis pemilihan *stack* yang mempertimbangkan kompromi antara efisiensi memori, modularitas kode, dan stabilitas performa sehingga dapat panduan yang relevan bagi entitas pengembang kecil menengah yang memiliki keterbatasan sumber daya. Dengan demikian, penelitian ini memperluas cakupan komparasi berbasis JavaScript sekaligus menyediakan bukti komprehensif bagi praktisi dan akademisi.

2. Metode Penelitian

Tahapan yang digunakan pada penelitian ini dapat dilihat pada Gambar 1. Penelitian dimulai dengan menganalisis literatur tentang *tech stack* berbasis JavaScript. Studi literatur menunjukkan terdapat tiga *stack* paling populer sebagai obyek komparasi: MEAN, MERN, dan MEVN. Langkah berikutnya ialah mengumpulkan data teknis yang relevan. Peneliti mengkaji prasyarat pengembangan aplikasi untuk masing-masing *stack*, pemahaman kerangka kerja *frontend–backend*, pengelolaan basis data MongoDB, pemanfaatan Node.js, serta menyelesaikan fase kontruksi aplikasi web obyek eksperimen.



Gambar 1. *Alur tahapan penelitian*

Perencanaan aplikasi diawali dengan identifikasi kebutuhan aplikasi sistem informasi yang akan dikembangkan di tahapan konstruksi. Kebutuhan tersebut meliputi fitur transaksional aktivitas kesiswaan sekolah yang dibatasi dalam lingkup dasar agar kompleksitas teknisnya tetap terkendali dan fokus komparasi tertuju pada perbandingan *stack* dalam rentang waktu penelitian yang sudah ditetapkan. Selanjutnya untuk memperkuat landasan teoretis, peneliti mengkaji literatur terkait komparasi *stack* dan evaluasi performa aplikasi, sehingga kerangka penelitian dapat didasarkan pada temuan-temuan empiris terdahulu.

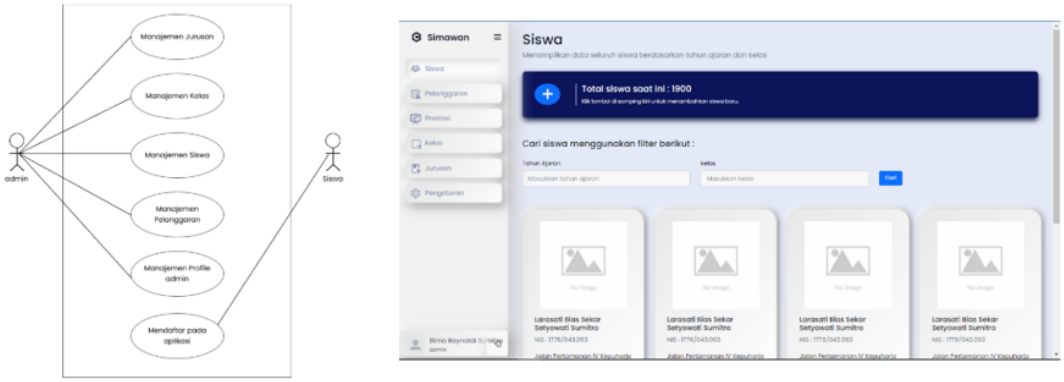
Tahap analisis kebutuhan aplikasi kemudian dilaksanakan dengan menyusun artefak seperti *use-case diagram* dan rancangan *user interface*. Rancangan tersebut menjadi pedoman pada fase konstruksi, di mana tiga aplikasi serupa dibangun menggunakan MEAN, MERN, dan MEVN secara terpisah. Metrik variabel yang digunakan pada tahap komparasi meliputi: (1) jumlah baris kode *(lines of code)*, (2) kinerja komputasi (*performance*), (3) prosedur eksekusi (*number of functions*), serta (4) penggunaan memori (*memory usage*).

Setelah ketiga aplikasi melewati fase konstruksi, peneliti melaksanakan pengujian terhadap ketiga set aplikasi untuk mengevaluasi kelebihan dan kekurangan setiap *stack*. Hasil pengujian dianalisis lebih lanjut dengan mengevaluasi ketiga set aplikasi berdasarkan metrik-metrik komparasi yang telah diuraikan di atas.

3. Hasil Penelitian

Obyek dan konfigurasi aplikasi

Proses komparasi dilakukan terhadap tiga set aplikasi dengan fitur transaksional yang identik sebagaimana dimodelkan dalam Gambar 2. Ketiga aplikasi dikembangkan pada basis infrastruktur yang sama, namun menggunakan versi *framework* yang berbeda pada fase konstruksi sebagaimana disajikan pada Tabel 1.



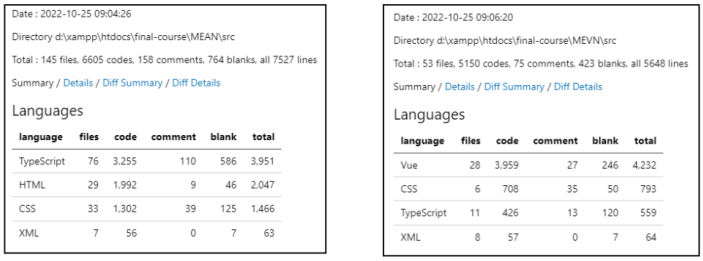
Gambar 2. *Fitur transaksional dan tampilan prototipe aplikasi obyek komparasi*

Tabel 1. *Versi konfigurasi stack obyek komparasi*

Konfigurasi MEAN *Stack*	Konfigurasi MERN Stack	Konfigurasi MEVN Stack
MongoDB version 6.2	MongoDB version 6.2	MongoDB version 6.2
ExpressJS version 4.16	ExpressJS version 4.16	ExpressJS version 4.16
Angular version 16	React version 18	VueJS version 3
NodeJS version 20	NodeJS version 20	NodeJS version 20
Chrome Desktop 122.0 64-bit	Chrome Desktop 122.0 64-bit	Chrome Desktop 122.0 64-bit

Jumlah baris kode (*lines of code, LOC*)

Metrik ini diukur dengan menganalisis kode ketiga aplikasi pada seluruh *layer* dan *package* yang telah dikembangkan pada fase konstruksi aplikasi dengan menggunakan ekstensi VS Code Counter pada editor Visual Studio Code sebagaimana ditunjukkan pada Gambar 3. Hasil komparasi metrik LOC dapat dilihat pada Tabel 2.

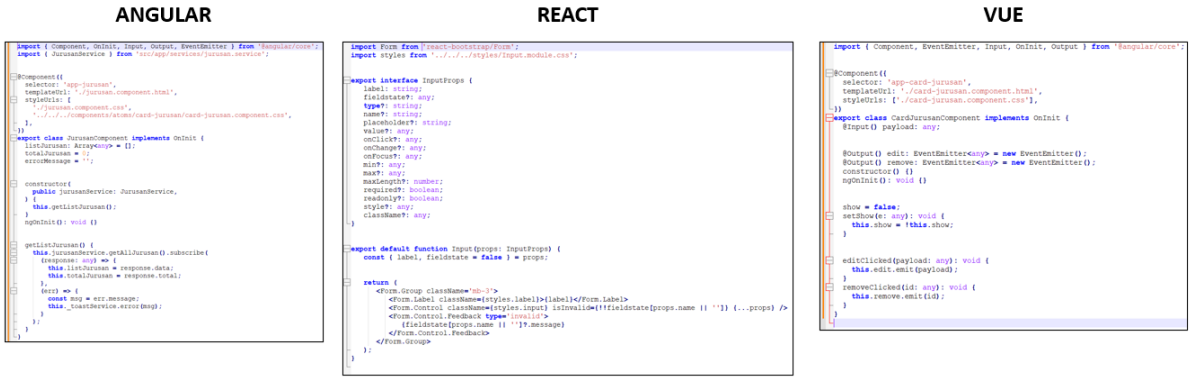


Gambar 3. *Tampilan layar ekstensi komparasi baris kode*

Tabel 2. *Rekapitulasi komparasi jumlah baris kode*

	Lines of Code						Total
Stack	**Typescript**	**CSS**	**XML**	**HTML**	**Typescript React**	**Vue**	
Angular	3225	1302	56	1993	-	-	**6605**
React	344	1021	60	-	3495	-	**4920**
Vue	426	708	57	-	-	3959	**5150**

Tabel 2 menunjukkan bahwa aplikasi berbasis Angular membutuhkan baris kode terbanyak (6605 LOC), lalu diikuti Vue (5150 LOC) dan React (4920 LOC).

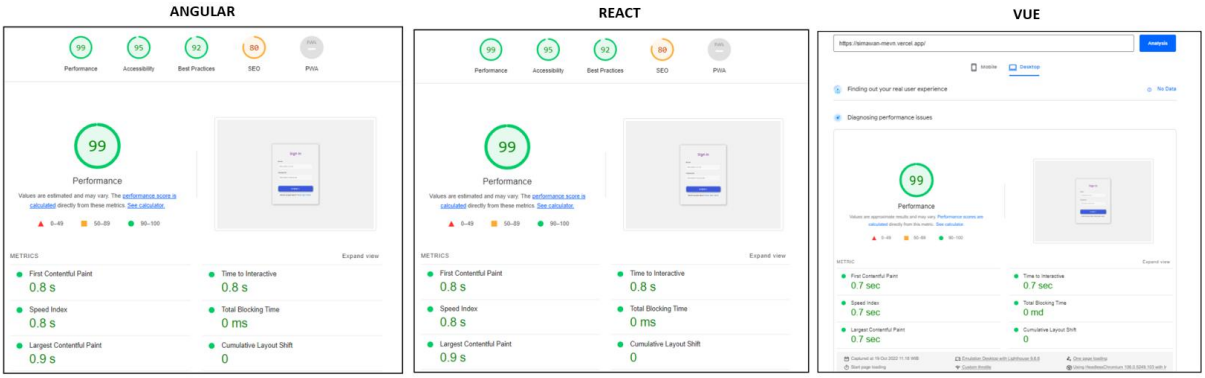


Gambar 4. *Contoh komparasi modularitas struktur kode Angular, React, dan Vue*

Hasil komparasi juga menunjukkan bahwa komposisi modularitas kode aplikasi bervariasi sebagaimana yang dicontohkan melalui Gambar 4. Angular memusatkan logika pada kode TypeScript (3225 LOC) dan memisahkan tampilan ke kode HTML secara terpisah (1993 LOC), sehingga struktur proyek lebih berlapis. React mengintegrasikan logika dan tampilan pada kode TypeScript React (3495 LOC), yang dapat menyederhanakan pemetaan fitur tetapi berpotensi menjadi titik utama ketika terjadi perubahan signifikan terhadap kode. Sementara itu, Vue menempatkan sebagian besar fungsionalitas dalam kode yang menggabungkan skrip, template, dan format *styling*, sehingga mengurangi jumlah kode tetapi membutuhkan pengelolaan struktur kode yang lebih rapi.

Kinerja aplikasi (*performance*)

Pengujian kinerja dilakukan menggunakan ekstensi Lighthouse pada aplikasi peramban Chrome Desktop 64-bit seperti ditunjukkan pada Gambar 5. Ketiga aplikasi telah di-*deploy* dengan basis data yang identik memanfaatkan skema *collections* pada MongoDB. Skema data yang digunakan merupakan data eksperimen yang digunakan pada fase konstruksi dan uji coba di infrastruktur pengembangan (*non-production*). Hasil komparasi menunjukkan skor kinerja yang hampir sama. Angular, React, dan Vue sama-sama memperoleh nilai *Performance* 99/100, *Accessibility* 95/100, dan *Best Practices* 92/100, sedangkan skor *SEO* tercatat 80/100 untuk Angular dan React. Hasil ini menandakan bahwa konfigurasi optimalisasi bawaan di ketiga *stack* mampu menghasilkan halaman yang responsif meski memuat data berkapasitas moderat.



Gambar 5. *Tampilan layar ekstensi Lighthouse untuk uji kinerja*

Perbandingan metrik juga mengungkap perbedaan minor tetapi konsisten: Vue unggul tipis dengan *First Contentful Paint* (FCP), *Speed Index*, dan *Time to Interactive* (TTI) sebesar 0,7 detik, sementara Angular dan React berada di kisaran 0,8 detik. *Largest Contentful Paint* (LCP) juga lebih

rendah pada Vue (0,7 detik) dibanding 0,9 detik pada dua *stack* lainnya. *Total Blocking Time* dan *Cumulative Layout Shift* sama-sama nol pada ketiga aplikasi, menandakan beban skrip minimal dan stabilitas tata letak baik.

Eksekusi fungsi (*call stack*)

Tabel 3 menunjukkan bahwa *stack* MERN memerlukan pemanggilan fungsi paling banyak yaitu sebanyak 27 kali untuk menyelesaikan empat operasi CRUD. Selanjutnya diikuti MEAN sebanyak 17 kali, sementara MEVN hanya 10 kali. Pola ini konsisten pada setiap kategori: MERN selalu memuncaki jumlah panggilan, MEAN berada di posisi selanjutnya, dan MEVN terendah. Dengan kata lain, MERN mengaktifkan rata-rata 6–7 fungsi per operasi, MEAN 2–7 fungsi, sedangkan MEVN cukup 2–3 fungsi. Selisih tersebut menggambarkan variasi komposisi logika: React cenderung memecah tanggung jawab ke dalam banyak fungsi pengelola (*handler*), Angular menggunakan layanan terpisah namun tetap mengandalkan beberapa fungsi terpadu, sedangkan Vue mengemas logika lebih ringkas dalam komponen *single-file*.

Tabel 3. *Komparasi eksekusi fungsi (call stack)*

Stack	*Create*	*Read*	*Update*	*Delete*	Total
MEAN (Angular)	5	3	7	2	**17**
MERN (React)	7	7	7	6	**27**
MEVN (Vue)	3	2	3	2	**10**

Dari sudut pandang pemeliharaan kode, jumlah pemanggilan fungsi yang lebih tinggi dapat memperjelas pemisahan tugas namun sekaligus menambah jejak *call stack* yang perlu ditelusuri saat proses *debug*. Sebaliknya, *call stack* yang lebih pendek pada MEVN memudahkan pelacakan alur eksekusi, namun berpotensi menggabungkan terlalu banyak logika di satu tempat jika tidak dikelola dengan baik. Berdasarkan hasil ini, pemilihan *stack* sebaiknya juga mempertimbangkan kriteria prioritas antara modularitas (MERN), keseimbangan struktur (MEAN), dan penyederhanaan *call stack* (MEVN) sesuai preferensi tim pengembang.

Konsumsi memori (*memory usage*)

Tabel 3 berikut menunjukkan bahwa React merupakan stack yang paling efisien dalam penggunaan memori, dengan rata-rata konsumsi 37260 kB. Angka tersebut sekitar 10% lebih rendah daripada Angular (41573 kB) dan 15% di bawah Vue (43958 kB). Pola penurunan memori pasca eksekusi fungsi juga terlihat paling tajam pada React, dari 44452 kB (P1) menjadi 33076 kB (P2), sebelum naik kembali ke 34252 kB (P3). Angular menunjukkan profil menengah: fluktuasi 8664 kB antara skenario tertinggi dan terendah, sedangkan Vue memerlukan memori terbesar tetapi paling stabil di mana terdapat selisih 3436 kB antar skenario.

Tabel 4. *Hasil komparasi konsumsi memori (memory usage)*

Stack / Percobaan	P1	P2	P3	Rata - rata
MEAN (Angular)	45.804k	37.140k	41.776k	41.573k
MERN (React)	44.452k	33.076k	34.252k	37.260k
MEVN (Vue)	46.408k	42.972k	42.496k	43.958k

Perbedaan ini mencerminkan strategi pengelolaan *state* dan mekanisme *garbage collection* pada masing-masing kerangka kerja. React cenderung agresif dalam melepaskan objek yang tak terpakai, menghasilkan konsumsi memori paling kecil namun terdapat lonjakan saat eksekusi awal (P1). Vue, dengan arsitektur *single-file*, mempertahankan lebih banyak konteks di memori sehingga stabil sepanjang transaksi. Angular berada di antaranya, menyeimbangkan modularitas layanan dan efisiensi memori.

4. Pembahasan

Hasil eksperimen menunjukkan bahwa Angular menghasilkan baris kode terbesar, React paling ringkas, dan Vue berada di antara keduanya. Penelitian [18] menegaskan bahwa volume kode cenderung meningkatkan kompleksitas pemeliharaan karena setiap revisi menyentuh lebih banyak artefak . Penelitian [19] turut melaporkan bahwa beban *debugging* lebih tinggi pada proyek dengan LOC besar. Dengan demikian, potensi pekerjaan pemeliharaan akan relatif lebih tinggi pada proyek aplikasi yang dikembangkan dengan Angular dibandingkan dengan React atau Vue.

Namun demikian, kedalaman *call-stack* memperlihatkan pola berbeda: React memanggil rata-rata 27 fungsi per operasi CRUD, sementara Vue hanya 10. Dalam penelitian [20], ditemukan bahwa modularitas ekstrem dapat meningkatkan keterlacakan tetapi juga memperpanjang jejak eksekusi program. Dengan demikian, meski React cenderung lebih efisien dari sisi LOC, namun kompleksitas *call-stack*-nya menuntut disiplin pengembang selama fase kontruksi dan pemeliharaan. Sebaliknya, Vue menawarkan alur eksekusi yang lebih singkat namun berpotensi menumpuk kode logika di satu komponen apabila tidak dikendalikan dengan baik. Perbedaan pola ini perlu dipertimbangkan oleh pengembang aplikasi ketika membandingkan aspek kemudahan pengembangan, skalabilitas, dan pekerjaan pemeliharaan.

Dari sisi performa, ketiga aplikasi meraih skor hampir identikdengan selisih FCP 0,1 detik antara Vue dan dua pesaingnya. Kinerja serupa ini menegaskan temuan [16] bahwa optimasi konfigurasi mampu meratakan kesenjangan performa antar kerangka kerja. Penelitian [10] dan [17] juga melaporkan perbedaan minor pada metrik *rendering* di lingkungan produksi. Secara praktis, selisih 0,1 hingga 0,2 detik semestinya tidak membawa dampak signifikan terhadap pengalaman pengguna. Pernyataan tersebut konsisten dengan yang ditemukan pada penelitian ini, namun dengan variabel lingkungan dan konfigurasi yang berbeda. Oleh karena itu, pilihan *stack* tidak hanya dipengaruhi oleh kinerja luaran halaman, melainkan juga faktor kualitas kode internal yang mempengaruhi siklus hidup proyek di fase selanjutnya.

Pengujian memori memperlihatkan React sebagai *stack* paling efisien, diikuti Angular dan Vue. Efisiensi ini mendukung rekomendasi [20] bahwa konsumsi memori yang efisien krusial bagi lingkungan dengan infrastruktur aplikasi yang terbatas. Vue menampilkan konsumsi stabil tapi dalam angka yang tinggi, di mana situasi tersebut relevan dengan kecenderungan arsitektur *single-file*. Angular berada di titik kompromi: memori moderat sekaligus struktur kode paling terpisah, sesuai praktik best-practice yang dianjurkan pada literatur [19]. Bagi proyek berskala kecil pada perangkat komputer dengan sumber daya terbatas, React menawarkan keuntungan pragmatis, sedangkan Vue cocok saat konsistensi performa lebih diutamakan. Angular dapat menjadi opsi yang wajar ketika tim membutuhkan struktur kode yang jelas tanpa kerugian memori yang signifikan.

Secara keseluruhan, hasil eksperimen komparasi pada penelitian ini mendukung temuan literatur bahwa tidak ada satu pun *stack* yang unggul secara absolut di semua metrik. Angular menawarkan arsitektur yang kokoh namun menghasilkan LOC besar. React menawarkan konsumsi memori dan LOC yang efisien tetapi eksekusi berlapis membawa kompleksitas tersendiri. Vue paling sederhana apabila ditelusuri berdasarkan strukturnya, namun relatif boros memori. Oleh karena itu, keputusan pemilihan *stack* harus mempertimbangkan konteks dan prioritas tujuan proyek: ketersediaan sumber daya, kebutuhan pemeliharaan jangka panjang, dan profil perangkat target.

5. Penutup

Hasil analisis komparasi mengungkap pola yang konsisten di keempat kriteria. Angular menghasilkan kode paling panjang dengan kompleksitas *call-stack* menengah. Sedangkan React menghasilkan kode paling ringkas, namun memecah logika kode ke dalam struktur fungsi yang paling kompleks. Vue berada di posisi tengah untuk jumlah baris kode namun memperlihatkan

eksekusi fungsi yang paling ringkas. Pada pengujian kinerja aplikasi, ketiganya meraih skor kinerja setara dan tidak menunjukkan perbedaan signifikan. Dari sisi konsumsi memori, React diindikasikan paling efisien, disusul dengan Angular dan Vue.

Temuan tersebut menegaskan bahwa tidak ada *stack* yang dominan unggul di semua aspek. Masing-masing *stack* menawarkan kriteria kompromi yang berbeda. React dapat digunakan pada aplikasi yang sedikit membutuhkan sumber daya komputasi dengan basis kode yang padat, selama tim pengembang dapat mengelola *call-stack* yang lebih rumit. Vue dapat dipilih ketika konsistensi performa dan eksekusi menjadi prioritas, meski dengan konsumsi memori yang sedikit lebih tinggi. Angular memberi struktur paling mandiri dengan arsitektur yang rapi, namun dengan jumlah baris kode terbesar. Dengan demikian, pemilihan *stack* sebaiknya mempertimbangkan konteks proyek, kapasitas tim, dan sumber daya perangkat keras.

6. Referensi

[1] Y. Gong, F. Gu, K. Chen, and F. Wang, "The Architecture of Micro-services and the Separation of Frond-end and Back-end Applied in a Campus Information System," in *2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications(AEECA)*, IEEE, Aug. 2020, pp. 321–324. doi: 10.1109/AEECA49918.2020.9213662.

[2] S. Supria and others, "Perbandingan Performa Framework Laravel, Flask API Python, dan PHP Native untuk Aplikasi API pada Data AIS Polbeng," in *Seminar Nasional Industri dan Teknologi*, 2024, pp. 17–23.

[3] S. I. Putri and M. Rofiq, "Perancangan Pemesanan Fasilitas Rumah Sakit Menggunakan Model View Controller (MVC) Berbasis Android," *SMATIKA JURNAL*, vol. 7, no. 02, pp. 36–39, Dec. 2017, doi: 10.32664/smatika.v7i02.156.

[4] A. S. Sari and R. Hidayat, "Designing Website Vaccine Booking System Using Golang Programming Language and Framework React JS," *Journal of Information System, Informatics and Computing*, vol. 6, no. 1, 2022, doi: 10.52362/jisicom.v6i1.760.

[5] S. Suwarno and A. P. Yulandi, "Analisis Performa Backend Framework: Studi Komparasi Framework Golang dan Node.js," *Jurasik*, vol. 8, no. 1, 2023, doi: 10.30645/jurasik.v8i1.551.

[6] J. S. Heredia and G. C. Sailema, "Comparative Analysis for Web Applications Based on REST Services: MEAN Stack and Java EE Stack," *KnE Engineering*, vol. 3, no. 9, p. 82, Dec. 2018, doi: 10.18502/keg.v3i9.3647.

[7] F. R. A. Abdullah, H. H. Nuha, R. G. Utomo, and A. D. Afasyah, "Implementation of User and Article Module Design and Testing on Innovation Dashboard Backend Using PHP Laravel and RESTful API," in *2024 International Conference on Decision Aid Sciences and Applications (DASA)*, IEEE, Dec. 2024, pp. 1–6. doi: 10.1109/DASA63652.2024.10836333.

[8] R. Annisa, P. A. Rahayuningsih, and A. Anna, "Aplikasi Kontrol Barang Habis Pakai Berbasis Web sebagai Solusi Manajemen Inventaris," *J-INTECH*, vol. 12, no. 02, pp. 411–421, Dec. 2024, doi: 10.32664/j-intech.v12i02.1544.

[9] P. Y. Pratama, Abd. C. Fauzan, and T. Prabowo, "Perancangan Sistem Informasi Inventaris pada PT. Rejoso Manis Indo Menggunakan Metode Rapid Application Development," *SMATIKA JURNAL*, vol. 14, no. 01, pp. 71–85, Jul. 2024, doi: 10.32664/smatika.v14i01.1209.

[10] G. Kaur and R. G. Tiwari, "Comparison and Analysis of Popular Frontend Frameworks and Libraries," in *2023 4th International Conference on Electronics and Sustainable Communication Systems (ICESC)*, 2023, pp. 1067–1073.

[11] A. Shukla, "Modern JavaScript Frameworks and JavaScript's Future as a FullStack Programming Language," *Journal of Artificial Intelligence & Cloud Computing*, pp. 1–5, Dec. 2023, doi: 10.47363/JAICC/2023(2)144.

[12] A. J. Poulter, S. J. Johnston, and S. J. Cox, "Using the MEAN stack to implement a RESTful service for an Internet of Things application," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, IEEE, Dec. 2015, pp. 280–285. doi: 10.1109/WF-IoT.2015.7389066.

[13] M. M. Ishaq, P. Singh, S. Badjatya, S. Kumar, Y. Tomar, and S. Bansal, "Design and Development of a User-Friendly Social Media App using the MERN Stack," in *2023 International Conference on Circuit Power and Computing Technologies (ICCPCT)*, IEEE, Aug. 2023, pp. 1730–1736. doi: 10.1109/ICCPCT58313.2023.10245371.
[14] M. R. Akbar, "Pengembangan Sistem Informasi Monitoring dan Evaluasi Perkuliahan Dengan Metode Agile Feature Driven pada Fakultas Teknik Universitas Negeri Jakarta," Universitas Negeri Jakarta, 2024. [Online]. Available: http://repository.unj.ac.id/45709/
[15] Y. Xing, J. Huang, and Y. Lai, "Research and Analysis of the Front-end Frameworks and Libraries in E-Business Development," in *Proceedings of the 2019 11th International Conference on Computer and Automation Engineering*, New York, NY, USA: ACM, Feb. 2019, pp. 68–72. doi: 10.1145/3313991.3314021.
[16] D. Bogusz, P. Ciszewski, and B. Pańczyk, "Performance analysis of web application client layer development tools us-ing Angular, React and Vue as examples," *Journal of Computer Sciences Institute*, vol. 32, pp. 223–230, Sep. 2024, doi: 10.35784/jcsi.6299.
[17] R. Vyas, "Comparative Analysis on Front-End Frameworks for Web Applications," *Int J Res Appl Sci Eng Technol*, vol. 10, no. 7, pp. 298–307, Jul. 2022, doi: 10.22214/ijraset.2022.45260.
[18] J. Samra, "Comparing Performance of Plain PHP and Four of Its Popular Frameworks," 2015. [Online]. Available: https://urn.kb.se/resolve?urn=urn:nbn:se:lnu:diva-45691
[19] R. Y. Endra, Y. Aprilinda, Y. Y. Dharmawan, and W. Ramadhan, "Analisis Perbandingan Bahasa Pemrograman PHP Laravel dengan PHP Native pada Pengembangan Website," *EXPERT: Jurnal Manajemen Sistem Informasi dan Teknologi*, vol. 11, no. 1, p. 48, Jun. 2021, doi: 10.36448/expert.v11i1.2012.
[20] S. Sharma and S. Srinivasan, "A Survey on Software Design Based and Project Based Metrics," *International Journal of Computer Theory and Engineering*, vol. 14, no. 2, pp. 54–61, 2022, doi: 10.7763/IJCTE.2022.V14.1310.
[21] W. Setiawan, "Studi Komparasi Pengembangan Website Menggunakan Framework dan Non Framework," in *Conference on Business, Social Sciences and Innovation Technology*, 2020, pp. 622–629.
[22] A. Niarman, Iswandi, and A. K. Candri, "Comparative Analysis of PHP Frameworks for Development of Academic Information System Using Load and Stress Testing," *International Journal Software Engineering and Computer Science (IJSECS)*, vol. 3, no. 3, pp. 424–436, Dec. 2023, doi: 10.35870/ijsecs.v3i3.1850.
[23] N. Nakajima, S. Matsumoto, and S. Kusumoto, "Jact: A Playground Tool for Comparison of JavaScript Frameworks," in *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*, IEEE, Dec. 2019, pp. 474–481. doi: 10.1109/APSEC48747.2019.00070.