Journal of Information Technology and Computer Science (INTECOMS)
Volume 8 Nomor 3, Tahun 2025
e-ISSN : 2614-1574
p-ISSN : 2621-3249

ANALISIS KOMPARATIF PERFORMA IMPLEMENTASI *LAZY LOADING DAN CODE SPLITTING* PADA *FRAMEWORK* REACT, VUE, DAN ANGULAR BERDASARKAN SKOR LIGHTHOUSE

COMPARATIVE ANALYSIS OF LAZY LOADING AND CODE SPLITTING IMPLEMENTATION PERFORMANCE ON REACT, VUE, AND ANGULAR FRAMEWORKS BASED ON LIGHTHOUSE SCORE

Alwan Alfian Setiawan[1], Esa Fauzi[2]
Program Studi S1 Teknik Informatika, Fakultas Teknik, Universitas Widyatama[1.2]
alwan.alfian@widyatama.ac.id[1], esa.fauzi@widyatama.ac.id[2]

ABSTRACT

Optimizing the performance of web applications is crucial to enhancing user experience and data access efficiency. Lazy Loading and Code Splitting are methods used to improve the performance of web applications in popular JavaScript frameworks: React, Vue, and Angular. This study aims to compare the performance of implementing these two techniques across the three frameworks using Lighthouse metrics. The research employs an experimental approach by building similar applications in each framework and measuring performance through First Contentful Paint (FCP), Largest Contentful Paint (LCP), Total Blocking Time (TBT), Cumulative Layout Shift (CLS), and Performance Score. The results indicate that React delivers the best performance with the fastest load times and superior layout stability. Vue offers component flexibility, while Angular maintains stability despite lower scores and longer load times. These findings provide optimization guidance for modern web application developers.

Keywords: Lazy Loading, Code Splitting, React, Vue, Angular, Google Lighthouse, web application performance.

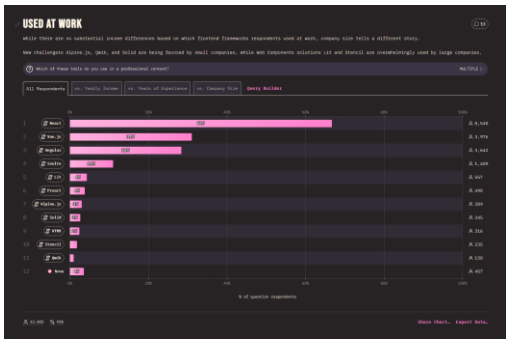
ABSTRAK

Optimasi performa aplikasi *web* sangat penting untuk meningkatkan pengalaman pengguna dan efisiensi akses data. Lazy Loading dan Code Splitting adalah salah satu metode yang digunakan untuk meningkatkan performa aplikasi *web* pada *framework* JavaScript populer: React, Vue, dan Angular. Tujuan penelitian ini adalah membandingkan performa implementasi kedua teknik tersebut di ketiga *framework* JavaScript tersebut menggunakan metrik Lighthouse. Metode penelitian menggunakan pendekatan eksperimental dengan membangun aplikasi serupa di masing-masing *framework* dan mengukur performanya menggunakan Google Lighthouse. Fokus pengukuran meliputi metrik *First Contentful Paint* (FCP), *Largest Contentful Paint* (LCP), *Total Blocking Time* (TBT), *Cumulative Layout Shift* (CLS), dan *Performance Score*. Hasil menunjukkan bahwa React memiliki performa terbaik dengan waktu muat tercepat dan unggul dalam stabilitas *layout*. Vue menawarkan fleksibilitas komponen yang baik, sedangkan Angular menunjukkan konsistensi performa meskipun dengan skor dan waktu muat yang lebih rendah. Temuan ini memberikan panduan bagi pengembang dalam memilih *framework* dan teknik optimasi untuk aplikasi *web* yang responsif dan efisien.

Kata Kunci: Lazy Loading, Code Splitting, React, Vue, Angular, Google Lighthouse, performa aplikasi web.

PENDAHULUAN

Laju perkembangan teknologi *web* saat ini berlangsung dengan sangat pesat, terutama dengan hadirnya beragam *framework* JavaScript yang memudahkan pembuatan aplikasi *web* interaktif dan dinamis (Dinh & Wang, 2020; Verma, 2022). Berdasarkan survei *State of JavaScript* untuk tahun 2024, React, Vue, dan Angular merupakan *framework* yang paling populer dan banyak digunakan oleh *web developer* di seluruh dunia (State of JavaScript, 2024).



Sumber: https://2024.stateofjs.com/en-US/libraries/front-end-frameworks/

Gambar 1. Data penggunaan *Framework* Javascript tahun 2024

Ketiga *framework* tersebut banyak digunakan untuk membuat *single-page application* (SPA) yang menawarkan *user experience* responsif dan efisien dibandingkan dengan aplikasi *web* biasa (Huotala et al., 2022).

Walaupun *Framework* JavaScript secara *default* sudah memiliki performa yang cukup baik namun performanya bisa saja menurun seiring dengan berkembangnya fitur dan kode pada aplikasi (Sinha & Sinha Jana, 2024). Oleh karena itu, optimasi performa aplikasi menjadi aspek yang sangat penting untuk memastikan pengalaman pengguna yang optimal dan responsif.

Salah satu teknik yang banyak diterapkan untuk menjaga performa aplikasi tetap optimal adalah *lazy loading*, yaitu strategi menunda pemuatan komponen atau sumber daya yang tidak langsung diperlukan saat halaman pertama kali diakses. Pendekatan ini memungkinkan aplikasi untuk tampil lebih cepat dan mengurangi waktu tunggu pengguna (Jain, 2022). Selain itu, teknik *code splitting* juga digunakan untuk membagi kode aplikasi menjadi bagian-bagian yang lebih kecil (*chunks*) dan dapat dimuat secara dinamis sesuai kebutuhan, sehingga beban awal aplikasi menjadi lebih ringan dan efisien (Aditya Kappagantula, 2025). Namun, *impact* yang dihasilkan dari implementasi kedua teknik tersebut tidak selalu sama antar *framework* sehingga akan memengaruhi performa dan *user experience* secara keseluruhan. Oleh karena itu *impact* implementasi teknik *lazy loading* dan *code splitting* pada *framework* JavaScript ini perlu dianasis lebih dalam untuk mengetahui kelebihan serta kekurangan teknik optimasi tersebut pada masing-masing *framework.*

Meskipun sejumlah penelitian sebelumnya telah membahas berbagai teknik optimasi performa pada aplikasi *web* menggunakan *lazy loading* dan *code splitting*, sebagian besar penelitian masih terfokus pada implementasi di satu *framework* tertentu tanpa melakukan perbandingan komprehensif antar *framework* utama seperti React, Angular, dan Vue (Jain, 2022; Mehmood, 2023). Selain itu, evaluasi performa umumnya masih berpusat pada metrik waktu muat halaman, sementara metrik *user experience* yang lebih komprehensif seperti *Total Blocking Time* (TBT) dan *Cumulative Layout Shift* (CLS) masih kurang diperhatikan (Tanudjaja & Tanone, 2021). Penelitian ini dibuat untuk mengisi kekosongan tersebut dengan melakukan analisis komparatif berbasis metrik performa dan *user experience* menggunakan alat evaluasi standar pada ketiga *framework* JavaScript tersebut.

Untuk menganalisis performa aplikasi *web*, Google Lighthouse dipilih sebagai alat evaluasi karena kemampuannya melakukan penilaian menyeluruh yang meliputi kecepatan muat, interaktivitas, dan stabilitas visual dalam satu platform terpadu dengan metrik yang terstandarisasi dan dapat diotomasi (Heričko et al., 2021). Selain itu, Lighthouse menyediakan skor performa yang mudah dipahami dan dapat digunakan secara konsisten dalam berbagai lingkungan pengujian, membuatnya sangat cocok untuk penelitian komparatif (Dubey et al., 2025). Walaupun ada *tools* lain seperti WebPageTest yang sangat mendalam dari segi analisis jaringan dan simulasi kondisi jaringan, namun kurang praktis untuk pengujian otomatis berulang dan tidak menyediakan skor kinerja agregat yang mudah diinterpretasi dalam konteks *user experiences* secara keseluruhan (Tutul Hossain et al., 2021). Sedangkan Chrome DevTools meskipun sangat berguna untuk *debugging* dan *profiling* secara manual, tidak menyediakan platform evaluasi otomatis dan terstruktur seperti Lighthouse, sehingga kurang ideal untuk studi kuantitatif komparatif (Chrome for Developers, 2024; Kuparinen, 2023).

Riset ini bertujuan untuk menganalisis dan membandingkan efektivitas implementasi teknik *lazy loading* dan *code splitting* pada beberapa *framework* JavaScript populer. Dengan demikian, diharapkan hasil penelitian ini dapat memberikan rekomendasi yang tepat bagi *web developer* dalam memilih teknik optimasi yang paling sesuai untuk meningkatkan *user experience* dari aplikasi yang sedang dikembangkan.

Lazy Loading

Lazy Loading adalah teknik pemuatan modul atau komponen secara dinamis hanya ketika dibutuhkan pada saat *runtime*, yang bertujuan mengurangi beban awal muatan aplikasi dan mempercepat waktu muat halaman pertama (*First Contentful Paint*) (Angular Documentations, 2025; Meta Open Source, 2025; Vue Router Documentations, 2025).

- **React** menggunakan *dynamic import()* dan *React.lazy/Suspense* untuk mengimple-mentasikan lazy loading komponen secara modular. Ketika komponen dibutuhkan, kode tersebut dimuat secara *asynchronous* (Meta Open Source, 2025).
- **Vue** mendukung *lazy loading* melalui *dynamic import()* pada Vue Router untuk komponen *route* tertentu, memuat modul hanya saat *route* diakses, serta fitur *Async Components* (Vue Router Documentations, 2025).
- **Angular** memiliki *lazy loading* berbasis modul yang terpisah dan dimuat secara dinamis dengan konfigurasi *routing* yang diatur di modul *root* dan modul fitur menggunakan *loadChildren* dan *bundling* modul secara otomatis (Angular Documentations, 2025).

Code Splitting

Code Splitting adalah teknik membagi *bundel* kode JavaScript besar menjadi bagian-bagian lebih kecil (*chunks*) yang dimuat secara terpisah, sehingga mempercepat pengiriman kode ke *browser* dan meningkatkan responsivitas aplikasi.

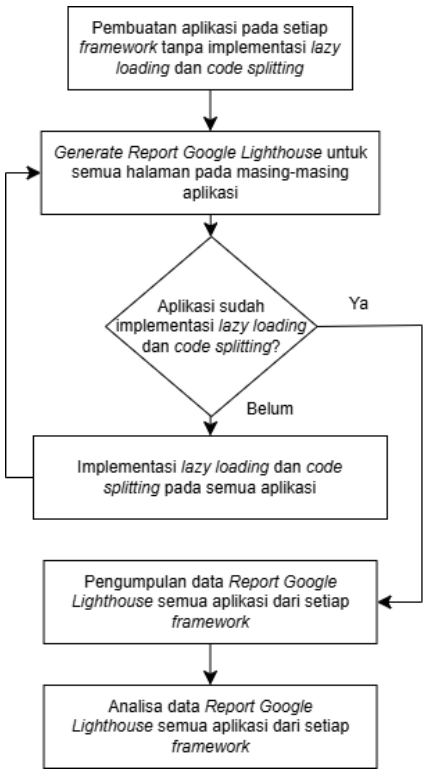
- **React** proses Code Splitting dilakukan secara otomatis dengan bundler seperti Webpack menggunakan *dynamic import()* atau manual dengan React *lazy() (Meta Open Source, 2025)*.
- **Vue** mengandalkan Webpack juga untuk membagi kode di level *route* dan komponen (Vue Router Documentations, 2025).
- **Angular** menggunakan Angular CLI dengan Webpack dan konfigurasi *built-in* untuk memecah bundel aplikasi ke dalam beberapa *chunk* yang dimuat secara dinamis (Angular Documentations, 2025).

Secara teknis, kedua teknik ini saling melengkapi untuk mengoptimalkan performa aplikasi *web modern* dengan menurunkan waktu loading awal dan meminimalkan *blocking resources* (Narender Reddy Karka, 2025).

METODE

Desain Penelitian

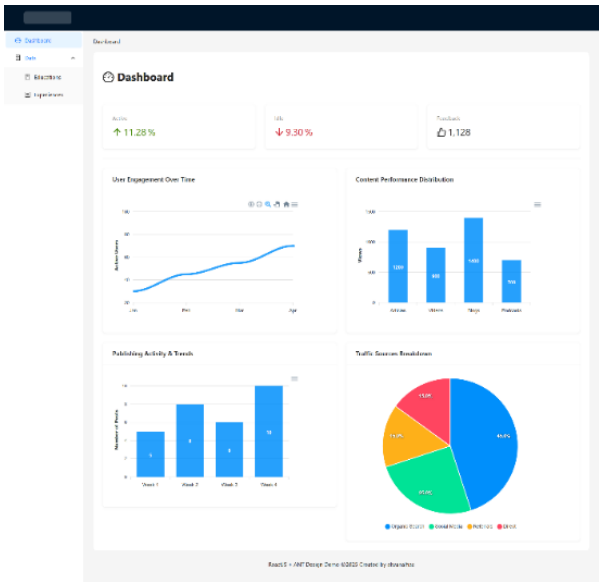
Penelitian ini menggunakan pendekatan kuantitatif dengan metode eksperimen untuk melakukan studi komparatif terhadap performa *framework* JavaScript *modern* dalam pengembangan aplikasi *web* (Aida et al., 2025; Syahrizal & Jailani, 2023). *Framework* yang dianalisis dalam penelitian ini adalah React, Vue, dan Angular yang dipilih berdasarkan popularitas dan penggunaannya yang luas dalam industri pengembangan *web* saat ini. Pendekatan eksperimen dipilih untuk memungkinkan pengukuran performa yang objektif dan terstandarisasi dari masing-masing *framework* dalam skenario pengembangan aplikasi *web* yang realistis (Khoirurrizal et al., 2024).



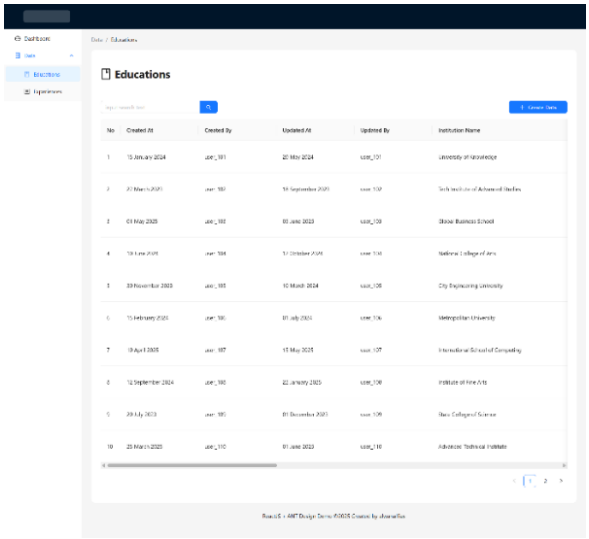
Gambar 2. Alur proses pengujian

Deskripsi Objek Penelitian

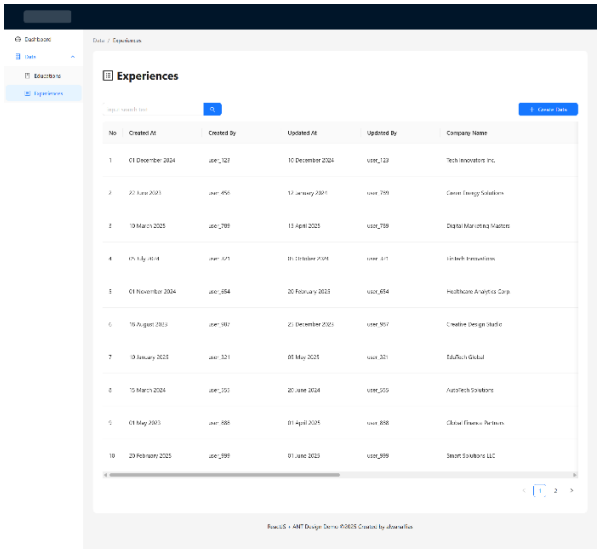
Penelitian ini dilakukan dengan mengembangkan sebuah aplikasi *web* sederhana yang dikembangkan menggunakan tiga *framework* JavaScript populer, yaitu React (versi 19.1.0), Vue (versi 3.5.15), dan Angular (versi 19.2.0) (Angular, 2025; React, 2025; Vue, 2025). Aplikasi yang dibuat memiliki tiga halaman utama: Dashboard, Educations, dan Experiences. Setiap halaman dirancang untuk menampilkan konten yang relevan dan interaktif, sehingga memungkinkan pengujian performa yang realistis pada masing-masing *framework* (Batoon & Calizon, 2025; Kryk & Plechawska-Wójcik, 2025). Semua komponen tampilan yang ada pada aplikasi menggunakan komponen dari Ant Design. Ant Design adalah sebuah *component library* yang membantu menjaga konsistensi tampilan antar aplikasi yang diuji, sehingga variabel tampilan tidak menjadi faktor pengganggu dalam pengukuran performa (Guo et al., 2023).



Gambar 3. Desain halaman Dashboard



Gambar 4. Desain halaman Educations



Gambar 5. Desain halaman Experiences

Teknik Pembuatan Aplikasi

Aplikasi dibuat secara terpisah pada masing-masing *framework* dengan struktur dan fitur yang seragam. Pada setiap *framework* akan dua versi aplikasi:

- **Versi 1** adalah aplikasi tanpa implementasi *code splitting* dan *lazy loading* (*baseline*).
- **Versi 2** adalah aplikasi seteah implementasi *code splitting* dan *lazy loading* (*optimized*).

Implementasi teknik *lazy loading* dan *code splitting* dilakukan sesuai dengan *best practice* pada masing-masing *framework*:

- **React:** Menggunakan *React.lazy* dan *Suspense* untuk *lazy loading*, serta *dynamic import* untuk *code splitting* (Meta Open Source, 2025).

```jsx
import React, { Suspense, lazy } from "react";
import { Route } from "react-router-dom";
import ErrorBoundary from "@components/errorBoundary";
import Loading from "@components/loading";

const Dashboard = lazy(() => import("@pages/dashboard"));

const RoutesApp = () => {
  return (
    <ErrorBoundary>
      <Suspense fallback={<Loading />}>
        <Route path="dashboard" element={<Dashboard />} />
      </Suspense>
    </ErrorBoundary>
  );
};

export default RoutesApp;
```

Gambar 6. Contoh implementasi *lazy loading* dan *code splitting* pada *React*

- **Vue**: Menggunakan *Vue Router* dengan *lazy loading* komponen dan *dynamic import* (Vue Router Documentations, 2025)

```jsx
import { createRouter, createWebHistory } from "vue-router";
import MainLayout from "./components/MainLayout.vue";

const routes = [
  {
    path: "/",
    component: MainLayout,
    children: [
      {
        path: "dashboard",
        component: () => import("./pages/Dashboard.vue"),
      },
    ],
  },
];

const router = createRouter({
  history: createWebHistory(),
  routes,
});

export default router;
```

Gambar 7. Contoh implementasi *lazy loading* dan *code splitting* pada *Vue*

- **Angular**: Menggunakan *Angular Router* dengan fitur *loadChildren* untuk *lazy loading* modul dan *code splitting* otomatis (Angular Documentations, 2025).

```jsx
import { Routes } from '@angular/router';

export const routes: Routes = [
  {
    path: 'dashboard',
    data: {
      breadcrumb: 'Dashboard',
    },
    loadComponent: () =>
      import('./pages/dashboard/dashboard.component').then(
        (m) => m.DashboardComponent
      ),
  },
];
```

Gambar 8. Contoh implementasi *lazy loading* dan *code splitting* pada *Angular*

Tujuan pembuatan dua versi ini adalah untuk membandingkan performa aplikasi sebelum dan sesudah penerapan teknik optimasi, sehingga dapat diukur seberapa besar dampak yang dihasilkan oleh *code splitting* dan *lazy loading*.

Konsistensi Antar Framework

Setiap versi aplikasi dikembangkan dengan menggunakan prinsip dan pola desain serupa, termasuk penggunaan *routing* halaman, pengelolaan *state* lokal dan global, serta pola komponen modular. Hal ini menjamin bahwa perbedaan performa yang diukur lebih berkaitan dengan karakteristik *inherent framework* dan teknik optimasi *lazy loading* dan *code splitting*, bukan pada perbedaan logika atau struktur kode aplikasi sendiri.

Pengujian Performa

Pengujian dilakukan pada lingkungan yang konsisten dengan spesifikasi sebagai berikut:

Tabel 1. Spesifikasi instrumen pengujian

Instrumen	Spesifikasi
Tester Tools	Google Lighthouse versi 12.6.0
Browser	Google Chrome versi 137.0.7151.104
Sistem Operasi	Windows 11 Pro, 64-bit
Hardware	Laptop dengan prosesor Intel Core i7-11370H, RAM 16GB
Jaringan	*Localhost, No Trottling*

Variabel dan Metrik yang Diukur

Pengukuran performa dilakukan menggunakan Google Lighthouse, sebuah alat audit performa yang memberikan skor berdasarkan berbagai metrik penting dalam konteks aplikasi web. Google Lighthouse dipilih karena kemampuannya memberikan evaluasi yang komprehensif dan objektif terhadap performa aplikasi web. Metrik yang diukur meliputi:

- *First Contentful Paint* (FCP): Waktu yang dibutuhkan untuk menampilkan konten pertama pada layar pengguna.
- *Largest Contentful Paint* (LCP): Waktu yang dibutuhkan untuk menampilkan elemen konten terbesar pada viewport.

- *Total Blocking Time* (TBT): Total waktu dimana thread utama diblokir sehingga tidak merespon input pengguna.
- *Cumulative Layout Shift* (CLS): Ukuran stabilitas *layout* halaman selama pemuatan.
- *Performance Score*: Skor keseluruhan performa aplikasi berdasarkan kombinasi metrik di atas.

Tabel 2. Sistem skor Google Lighthouse

Parameter	Contoh Nilai	Satuan
FCP	0.7	Detik (s)
LCP	0.8	Detik (s)
TBT	100	Milidetik (ms)
CLS	0	Skor (0-1)
Speed Index	1,2	Detik (s)
Performance Score	98	Skor (0-100)

Prosedur Pengumpulan dan Analisis Data

Pengumpulan data dilakukan dengan menguji semua halaman web dari setiap aplikasi yang dibuat pada masing-masing *framework* dengan Google Lighthouse. Setiap halaman akan diuji berulang banyak 5 kali. Hasil akhir dari pengujian setiap halaman dihitung dari rata-rata hasil pengujian. Data yang didapatkan akan dimasukan dalam tabel untuk selanjutnya dapat dilakukan analisis (Correia et al., 2021).

Pada proses analisis, data performa antara aplikasi versi 1 dan versi 2 akan dibandingkan untuk mendapatkan *impact* yang terjadi dengan implementasi *lazy loading* dan *code splitting* pada setiap *framework* (Tanudjaja & Tanone, 2021). Seluru data *impact* ini kemudian akan dibandingkan untuk mendapatkan kesimpulan.

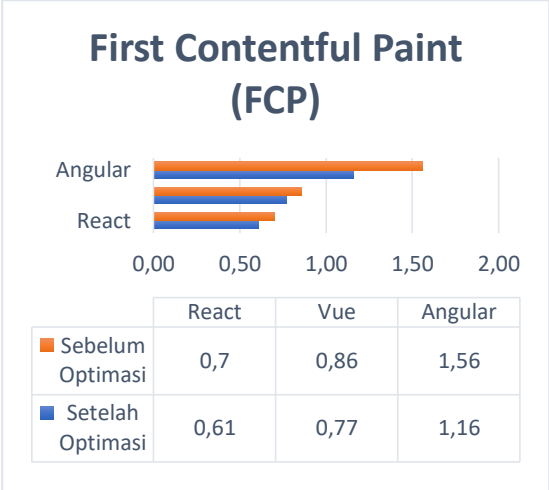
Semua aplikasi yang diuji dibuat dalam lingkungan pengembangan yang serupa untuk menjamin keseragaman hasil. Pengujian dilaksanakan pada perangkat keras dan perangkat lunak yang serupa, termasuk versi terbaru dari *browser* dan kerangka kerja, untukmengurangivariabel luar yang mungkin memengaruhi hasil (Tripon et al., 2021).

Batasan Penelitian

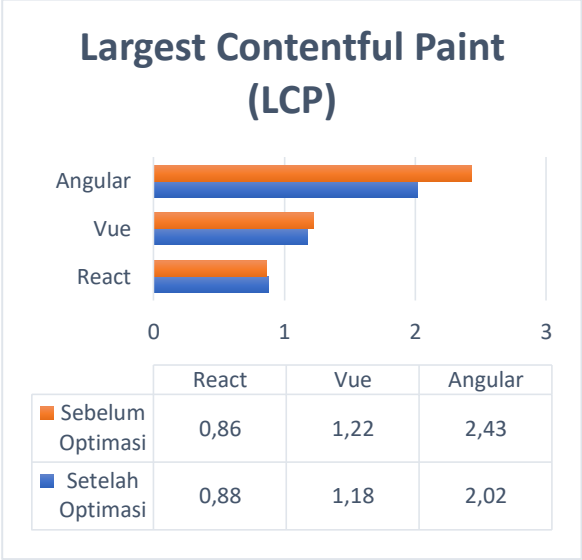
- Penelitian ini hanya menguji aplikasi sederhana dengan tiga halaman, sehingga hasil mungkin berbeda pada aplikasi yang lebih kompleks.
- Pengujian dilakukan pada perangkat dan jaringan tertentu, sehingga hasil performa dapat bervariasi pada kondisi perangkat dan jaringan lain.
- Fokus penelitian adalah pada metrik performa yang diukur oleh Google Lighthouse, tanpa mempertimbangkan aspek lain seperti keamanan atau skalabilitas.

HASIL DAN PEMBAHASAN

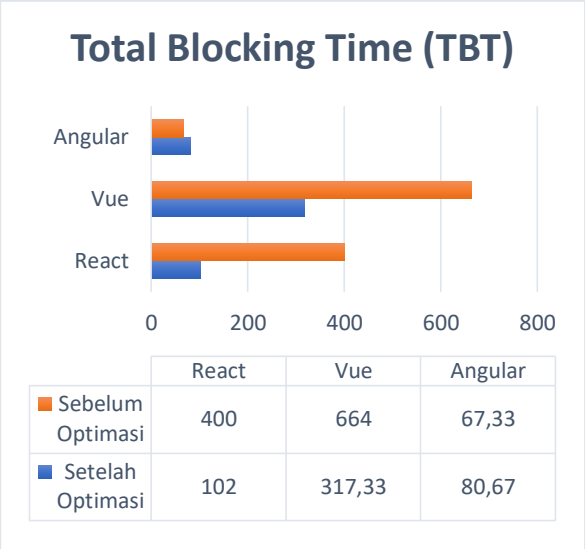
Dari uji performa dengan menggunakan Google Lighthouse pada setiap aplikasi yang sudah dibuat, didapatkan hasil sebagai berikut.



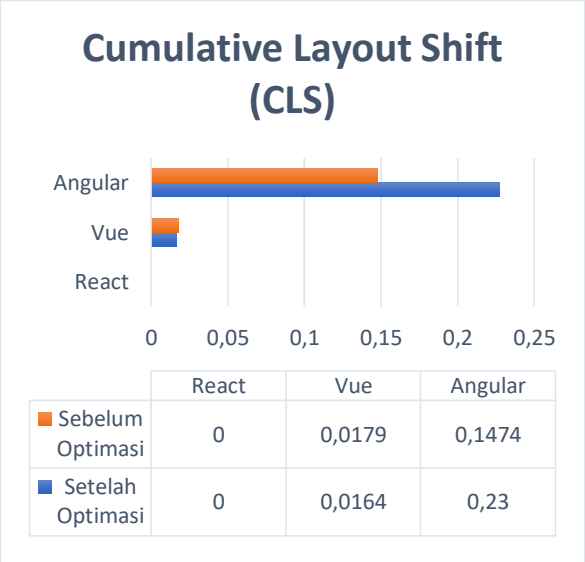
Gambar 9. Hasil komparasi FCP setelah optimasi



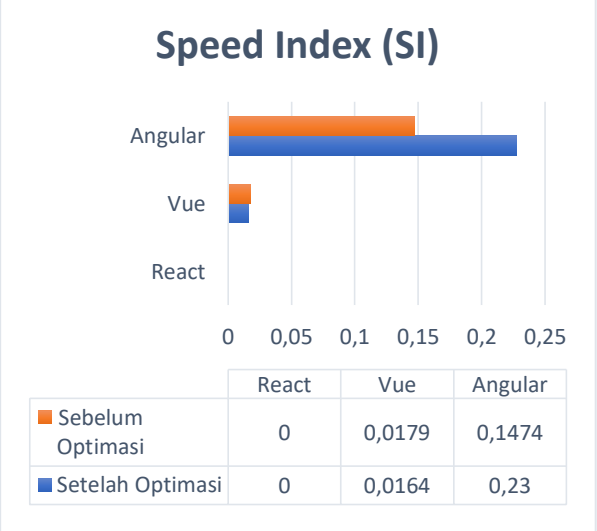
Gambar 10. Hasil komparasi LCP setelah optimasi



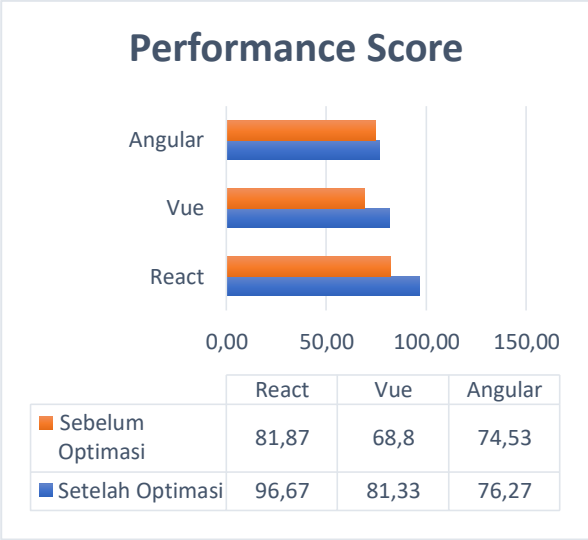
Gambar 11. Hasil komparasi TBT setelah optimasi



Gambar 12. Hasil komparasi FCP setelah optimasi



Gambar 13. Hasil komparasi FCP setelah optimasi



Gambar 14. Hasil komparasi FCP setelah optimasi

Jika seluruh data performa dari aplikasi versi 1 dan versi 2 dibandingkan, maka akan didapatkan data *impact* seperti berikut

Tabel 3. Tabel hasil uji performance tanpa optimasi

Metrik	React	Vue	Angular
FCP	0.7	0.86	1.56
LCP	0.86	1.22	2.43
TBT	400	664	67.33
CLS	0	0.018	0.15
Speed Index	1.27	1.99	1.56
Performance Score	81.87	68.8	74.53

Tabel 4. Tabel hasil uji performance setelah optimasi

Metrik	React	Vue	Angular
FCP	0.61	0.77	1.16
LCP	0.88	1.18	2.02
TBT	102	317.33	80.67
CLS	0	0.016	0.23
Speed Index	1.11	2.39	1.34
Performance Score	96.67	81.33	76.27

Tabel 5. Data *impact* setelah implementasi *code splitting* dan *lazy loading*

Framework	Metrik	Perubahan(%)
React	FCP	-12.86%
	LCP	-2.33%
	TBT	-75.5%
	CLS	0%
	SI	-12.60%
	Performance	+18.01%

Vue	FCP	-10.47%
	LCP	-3.28%
	TBT	-57.17%
	CLS	-10.61%
	SI	-20.10%
	Performance	+18.18%
Angular	FCP	-25.64%
	LCP	-16.46%
	TBT	+19.9%
	CLS	+56.46%
	SI	-14.0%
	Performance	+1.13%

Hasil uji performa ketiga *framework* populer, yaitu React, Vue, dan Angular, menunjukkan bahwa penerapan teknik optimasi *lazy loading* dan *code splitting* memberikan dampak yang berbeda-beda pada masing-masing *framework*, yang dipengaruhi oleh karakteristik teknis dan arsitektur internal mereka.

React menunjukkan peningkatan performa yang paling signifikan dengan penurunan *First Contentful Paint* (FCP) sebesar 12,86% serta *Total Blocking Time* (TBT) turun drastis hingga 74,5%. Selain itu, skor performa keseluruhan meningkat hingga 18,04%. Keunggulan ini dapat dijelaskan dengan teknologi Virtual DOM yang digunakan oleh React yang secara efisien mengelola proses *rendering* ulang antar komponen sehingga beban *runtime* dapat diminimalisir. Optimasi *lazy loading* dan *code splitting* lebih efektif berjalan pada React karena struktur aplikasi yang ringan dan konsisten, sehingga pengurangan waktu *blocking* dan waktu muat langsung berdampak besar untuk keseluruhan *user experience*.

Vue juga tampil dengan peningkatan signifikan, khususnya pada penurunan TBT sebesar 52,17% dan FCP sebesar 10,47%, yang menunjukkan aplikasi lebih responsif dan cepat dimuat setelah optimasi. Namun, peningkatan metrik *Speed Index* (SI) sebesar 20,1% menandakan bahwa meskipun waktu muat utama berkurang, beberapa elemen visual belum optimal saat halaman di-*render* dengan cepat. Hal ini bisa dipengaruhi oleh penggunaan komponen dinamis atau mekanisme *rendering* Vue yang memerlukan penyesuaian tambahan untuk mencapai kinerja *rendering* yang mulus. Secara keseluruhan, Vue tetap menunjukkan efisiensi yang baik dan potensi untuk peningkatan lebih lanjut.

Angular hasilnya lebih kompleks dengan penurunan FCP sebesar 25,64% dan LCP sebesar 16,46%, menunjukkan waktu muat utama yang membaik secara signifikan setelah optimasi. Namun, *Total Blocking Time* (TBT) meningkat 19,9%, dan *Cumulative Layout Shift* (CLS) meningkat tajam hingga 56,46%. Hal ini menggambarkan adanya *overhead runtime* tambahan dan kompleksitas halaman yang menyebabkan gangguan interaktivitas dan stabilitas visual. Meskipun Angular memiliki fitur bawaan yang kuat, seperti *dependency injection* dan banyaknya fitur *runtime*, hal ini justru bisa menjadi beban saat melakukan optimasi, terutama jika *code splitting* dan *lazy loading* tidak diatur dengan sangat hati-hati. Skor performa keseluruhan meningkat tipis hanya 1,13%, menandakan bahwa optimasi perlu difokuskan pada pengurangan *overhead runtime* dan perbaikan *layout* saat perubahan elemen halaman

Meskipun Angular menunjukkan penurunan yang signifikan di sebagian metrik utama seperti FCP dan LCP, kenaikan TBT dan CLS menjadi perhatian serius. Lonjakan TBT menandakan adanya peningkatan pada waktu *blocking*, kemungkinan akibat *overhead runtime* dan kompleksitas modul yang meningkat setelah optimasi. CLS yang naik tajam menunjukkan masalah stabilitas visual terkait dengan perubahan layout atau pemuatan ulang elemen yang tidak optimal akibat *code splitting* yang kurang efisien. Hal ini menunjukkan bahwa optimasi pada Angular memerlukan pendekatan lebih hati-hati dan spesifik untuk mengatasi beban *runtime* dan *layout* dinamis, agar tidak mengorbankan *user experience*. Skor performa keseluruhan yang hanya sedikit meningkat menegaskan bahwa optimasi pada Angular belum memberikan dampak besar tanpa perbaikan sampai ke tingkat konfigurasi internal.

KESIMPULAN

Berdasarkan hasil uji coba dan analisis data performa dari metrik Google Lighthouse, dapat

disimpulkan bahwa secara umum teknik optimasi *lazy loading* dan *code splitting* berhasil meningkatkan performa aplikasi *web*, terutama dalam pengurangan waktu *blocking* dan waktu muat utama, yang berdampak positif pada interaktivitas dan pengalaman pengguna. Namun, hasil yang bervariasi pada Angular menunjukkan bahwa setiap framework memerlukan pendekatan optimasi yang disesuaikan dengan arsitektur dan kompleksitasnya masing-masing.

Untuk React dan Vue, optimalisasi bisa berfokus pada peningkatan lebih lanjut pada responsivitas tampilan dan pengelolaan komponen dinamis agar *Speed Index* lebih optimal. Sedangkan untuk Angular, penting untuk meminimalkan *overhead runtime* dan mengelola perubahan *layout* agar stabilitas visual tetap terjaga.

Dari hasil yang diperoleh, disarankan agar para *web developer* dan peneliti fokus pada evaluasi komprehensif serta penerapan teknik optimasi yang sesuai dengan karakteristik *framework* yang digunakan, agar performa aplikasi *web* dapat dioptimalkan tanpa mengorbankan kenyamanan dan *user experience*.

DAFTAR PUSTAKA

Aditya Kappagantula. (2025). Optimizing JavaScript application performance: A comprehensive guide. *International Journal of Science and Research Archive*, *14*(1), 1279–1303. https://doi.org/10.30574/ijsra.2025.14.1.0234

Aida, A., Hermina, D., & Norlaila, N. (2025). JENIS DATA PENELITIAN KUANTITATIF (Korelasional, Komparatif, Dan Eksperimen). *Al-Manba Jurnal Ilmiah Keislaman Dan Kemasyarakatan*, *10*(1), 31–40. https://doi.org/10.69782/almanba.v10i1.48

Angular. (2025). *Angular versioning and releases*. AngularJS Documentation. https://angular.dev/reference/releases

Angular Documentations. (2025). *Lazy-loading feature module*. Angular. https://v17.angular.io/guide/lazy-loading-ngmodules

Batoon, J. A., & Calizon, J. M. (2025). Comparative Review for the Edges among JS Frameworks: Angular vs React in Web Application Development. *International Journal of Multidisciplinary: Applied Business and Education Research*, *6*(5), 2386–2400. https://doi.org/10.11594/ijmaber.06.05.23

Chrome for Developers. (2024). *Analyze runtime performance*. Google.

Correia, F., Ribeiro, O., & Silva, J. C. (2021). Progressive Web Apps Development: Study of Caching Mechanisms. *2021 21st International Conference on Computational Science and Its Applications (ICCSA)*, 181–187. https://doi.org/10.1109/ICCSA54496.2021.00033

Dinh, D., & Wang, Z. (2020). *MODERN FRONT-END WEB DEVELOPMENT-How libraries and frameworks transform everything*.

Dubey, V., Bhimrao, B., & Verma, S. (2025). Evaluating The Digital Performance and Accessibility of IIT Library Websites Using Google Lighthouse. *Article in International Journal of Computer Sciences and Engineering*, *13*(3), 16–23. https://doi.org/10.26438/ijcse/v13i3.1623

Guo, Z., Lv, S., Li, J., Luo, X., Ji, Q., Zhang, H., Chen, L., & Ma, H. (2023). Design and implementation of transformer state sensing and predictive operation and maintenance system interface based on Ant Design Vue. *2023 3rd International Conference on Electrical Engineering and Mechatronics Technology (ICEEMT)*, 233–237. https://doi.org/10.1109/ICEEMT59522.2023.10263256

Heričko, T., Šumak, B., & Brdnik, S. (2021). *Towards Representative Web Performance Measurements with Google Lighthouse*. 39–42. https://doi.org/10.18690/978-961-286-516-0.9

Huotala, A., Luukkainen, M., & Mikkonen, T. (2022). Benefits and Challenges of

Isomorphism in Single-page Applications: Case Study and Review of Gray Literature. *Journal of Web Engineering*, *21*(8), 2363–2404. https://doi.org/10.13052/jwe1540-9589.2186

Jain, V. (2022). OPTIMIZING WEB PERFORMANCE WITH LAZY LOADING AND CODE SPLITTING. *International Journal of Core Engineering & Management*, *7*. https://doi.org/10.5281/zenodo.14956631

Khoirurrizal, M. F., Hidayat, C. R., & Ruuhwan, R. (2024). ANALISIS PERBANDINGAN FRAMEWORK FRONT-END JAVASCRIPT SOLIDJS DAN VUEJS PADA PENGEMBANGAN WEBSITE INTERAKTIF. *Jurnal Informatika Dan Teknik Elektro Terapan*, *12*(2). https://doi.org/10.23960/jitet.v12i2.4106

Kryk, J., & Plechawska-Wójcik, M. (2025). Multi-aspect comparative analysis of JavaScript programming frameworks – React.js and Solid.js. *Journal of Computer Sciences Institute*, *34*, 68–75. https://doi.org/10.35784/jcsi.6712

Kuparinen, S. (2023). *Improving Web Performance by Optimizing Cascading Style Sheets (CSS): Literature Review and Empirical Findings*. http://www.cs.helsinki.fi/

Mehmood, A. (2023). WEB DEVELOPMENT FRAMEWORKS: COMPARING REACT, ANGULAR, AND VUE. *Computer Science Bulletin*.

Meta Open Source. (2025). *lazy*. React. https://react.dev/reference/react/lazy

Narender Reddy Karka. (2025). Front-End Performance Optimization: A Comprehensive Guide. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, *11*(2), 83–100. https://doi.org/10.32628/CSEIT251112389

React. (2025). *React Versions*. Meta Open Source. https://react.dev/versions

Sinha, K., & Sinha Jana, D. (2024). *The Role of JavaScript Frameworks in Performance Optimization: A Comparative Study* (Vol. 10).

State of JavaScript. (2024). *FRONT-END FRAMEWORKS*. https://2024.stateofjs.com/en-US/libraries/front-end-frameworks/

Syahrizal, H., & Jailani, M. S. (2023). Jenis-Jenis Penelitian Dalam Penelitian Kuantitatif dan Kualitatif. *Jurnal QOSIM Jurnal Pendidikan Sosial & Humaniora*, *1*(1), 13–23. https://doi.org/10.61104/jq.v1i1.49

Tanudjaja, D., & Tanone, R. (2021). Analisis Penerapan Code Splitting Library React pada Aplikasi Penjualan Mebel Berbasis Website. *Jurnal Teknik Informatika Dan Sistem Informasi*, *7*(2). https://doi.org/10.28932/jutisi.v7i2.3493

Tripon, T.-D., Adela Gabor, G., & Valentina Moisi, E. (2021). Angular and Svelte Frameworks: a Comparative Analysis. *2021 16th International Conference on Engineering of Modern Electric Systems (EMES)*, 1–4. https://doi.org/10.1109/EMES52337.2021.9484119

Tutul Hossain, Md., Hassan, R., Amjad, M., & Rahman, Md. A. (2021). Web Performance Analysis: An Empirical Analysis of E-Commerce Sites in Bangladesh. *International Journal of Information Engineering and Electronic Business*, *13*(4), 47–54. https://doi.org/10.5815/ijieeb.2021.04.04

Verma, D. (2022). A Comparison of Web Framework Efficiency– Performance and network analysis of modern web frameworks. *Theseus*.

Vue. (2025). *Releases*. VueJS Documentation. https://vuejs.org/about/releases

Vue Router Documentations. (2025). *Lazy Loading Router*. Vue Router. https://router.vuejs.org/guide/advanced/lazy-loading.html