SNESTIK

Seminar Nasional Teknik Elektro, Sistem Informasi, dan Teknik Informatika

https://ejurnal.itats.ac.id/snestik dan https://snestik.itats.ac.id

Informasi Pelaksanaan:
SNESTIK IV - Surabaya, 27 April 2024
Ruang Seminar Gedung A, Kampus Institut Teknologi Adhi Tama Surabaya

Informasi Artikel
DOI : 10.31284/p.snestik.2024.5878

Prosiding ISSN 2775-5126

Fakultas Teknik Elektro dan Teknologi Informasi-Institut Teknologi Adhi Tama Surabaya
Gedung A-ITATS, Jl. Arief Rachman Hakim 100 Surabaya 60117 Telp. (031) 5945043
Email : snestik@itats.ac.id

Optimasi Performa React JS Menggunakan Code Splitting

Fauzan Pradana, Dika Rizky Yunianto, Sofyan Noor Arief

Politeknik Negeri Malang

e-mail: fauzanpr.06@gmail.com

ABSTRACT

Performance optimization in web programming has become a primary focus due to user demands for faster loading times. Code splitting, which divides JavaScript code into smaller pieces to be loaded on demand, is one effective approach. In the context of React, built-in features such as React.Lazy and React.Suspense enable code splitting down to the component level. Previous research has shown that implementing code splitting can significantly reduce client-side rendering time and speed up the loading time of React JS-based web applications. The aim of this study is to investigate the impact of code splitting on the performance of React JS web applications. The results indicate that code splitting indeed improves the performance of React JS-based web applications, as measured using Lighthouse.

Keywords: *React JS; Code splitting; Web performance.*

ABSTRAK

Optimasi kinerja dalam pemrograman web telah menjadi fokus utama karena permintaan akan waktu muat yang lebih cepat oleh pengguna. *Code splitting*, yang membagi kode JavaScript menjadi potongan-potongan kecil untuk dimuat sesuai kebutuhan, adalah salah satu pendekatan yang efektif. Dalam konteks *React*, fitur bawaan seperti *React.Lazy* dan *React.Suspense* memungkinkan *code splitting* hingga tingkat komponen. Penerapan *code splitting* dapat mengurangi waktu *rendering* di sisi klien secara signifikan dan mempercepat waktu pembuatan aplikasi web berbasis *React JS*. Hal ini dapat dilihat dari skor hasil pengukuran aplikasi web React JS sebelum menggunakan *code splitting* dengan rerata waktu *First Contentful Paint* sebesar 7,49 detik sedangkan sesudah menggunakan *code splitting* menjadi sebesar 1,98 detik. Efisiensi waktu juga dapat dilihat dari hasil pengukuran skor *Lighthouse*, dimana skor rerata sebelum menggunakan *code splitting* sebesar 54,9 sedangkan setelah menggunakan *code splitting* sebesar 64,6.

Kata kunci: *React JS*; *Code splitting*; Performa web.

PENDAHULUAN

Perhatian terhadap optimasi kinerja dalam pengembangan web telah meningkat secara signifikan, dikarenakan tuntutan pengguna akan pengalaman *browsing* yang cepat dan responsif. Dalam menghadapi tantangan ini, diperlukan strategi dan teknik yang dapat meningkatkan waktu muat halaman serta kinerja keseluruhan aplikasi. Optimasi kinerja dalam pemrograman web telah mendapat perhatian yang signifikan karena pengguna menuntut waktu muat yang lebih cepat. Teknik pengoptimalan kinerja web seperti *lazy loading*, *code splitting*, dan *Content Delivery Networks* (CDN) telah diadopsi secara luas [1]. Cara melakukan optimasi kinerja antara lain dengan mengeliminasi *redundant computation* dan mengoptimasi performa aplikasi. Teknik seperti *multithreading* dapat diimplementasikan di semua aplikasi terlepas dari *framework* yang digunakan [2].

Sebuah *website* dapat dikembangkan dengan menggunakan *framework React JS*. *React JS* adalah sebuah *library open-source* yang digunakan untuk mengembangkan *user interface* khususnya untuk aplikasi berbasis satu halaman (*single page application*). *React* memungkinkan para pengembang untuk membuat aplikasi web yang dapat menggunakan data dan berubah tanpa perlu melakukan *refresh* halaman [3].

Dalam hal peningkatan performa, *code splitting* merupakan salah satu aspek kunci yang sangat berperan dalam mengoptimalkan ukuran bundel dan mempercepat waktu pemuatan, bahkan pada aplikasi web yang sangat kompleks [1]. *Code splitting* dikenal sebagai salah satu pendekatan yang dapat digunakan untuk mengoptimalkan kinerja aplikasi. *Code splitting* adalah metode yang membagi kode JavaScript menjadi potongan-potongan yang lebih kecil, yang sering disebut sebagai "*chunks*". *Chunks* ini hanya dimuat ke *browser* ketika diperlukan sehingga dapat mengurangi beban awal pada saat memuat halaman pertama kali (*initial render*) dalam aplikasi web [4].

Dalam konteks *React*, *code splitting* bisa dilakukan dengan memanfaatkan fitur yang disediakan oleh *framework* tersebut. *React* mendukung *code splitting* hingga tingkat komponen dengan menyediakan *method* bawaan yang disebut *React.Lazy* [5]. *Method* ini memungkinkan pengembang untuk memuat komponen React secara dinamis ke browser hanya ketika diperlukan. Selain itu, *suspense* adalah fitur lain yang tersedia dalam *React.Suspense* memungkinkan penggunaan tampilan pengganti (*fallback*) selama proses memuat komponen sehingga pengalaman pengguna tetap terjaga hingga komponen selesai dimuat [5].

Code splitting merupakan metode yang efektif dalam mengoptimalkan performa aplikasi berbasis web, dibuktikan dari penelitian sebelumnya yang telah dilakukan. Salah satu penelitian yang relevan berjudul *"Analisis Penerapan Code Splitting Library React pada Aplikasi Penjualan Mebel Berbasis Website"* di mana penerapan *code splitting* pada sebuah situs web penjualan mebel berhasil mengurangi waktu *rendering* di sisi klien [6]. Penelitian lain yang berjudul *"Analyzing the Efficiency and Performance Optimization Techniques of React.Js in Modern Web Development"* membahas teknik-teknik optimasi performa dan efisiensi dalam pengembangan aplikasi web berbasis *React JS*. Hasil penelitian tersebut menegaskan bahwa *code splitting* merupakan salah satu aspek kunci yang sangat berperan dalam mengoptimalkan ukuran bundel dan mempercepat waktu pemuatan, bahkan pada aplikasi web yang sangat kompleks [1]. Terdapat juga penelitian lain yang relevan dengan judul *"Methods of Improving and Optimizing React Web- applications"*, juga mengadopsi *code splitting* sebagai salah satu teknik yang berhasil meningkatkan performa aplikasi berbasis *React JS* [4].

Dalam konteks pengembangan *website* menggunakan *React JS*, penting untuk mempertimbangkan berbagai strategi untuk meningkatkan performa. Oleh karena itu, pada jurnal ini diusulkan penggunaan *code splitting* untuk meningkatkan performa React JS pada suatu website. Penerapan teknik *code splitting*, komponen-komponen dalam aplikasi *React* dapat dibagi menjadi *bundle-bundle* terpisah sehingga hanya komponen yang diperlukan untuk

halaman tertentu yang dimuat, mengurangi waktu muat halaman dan mempercepat waktu respon. Diharapkan bahwa dengan menerapkan teknik ini, performa kinerja suatu *website* yang menggunakan *React* dapat meningkat secara signifikan, memberikan pengalaman pengguna yang lebih responsif dan menyenangkan.

METODE

Dalam melakukan optimasi *React JS*, pada jurnal ini menggunakan sebuah *website* yang dirancang khusus untuk memfasilitasi kolaborasi antara dosen dan mahasiswa. Tujuan utama dari *website* tersebut adalah untuk meningkatkan kompetensi mahasiswa, memperkaya portofolio dan mendalami pengetahuan sesuai dengan bidang riset para dosen.

React JS

Jurnal ini menggunakan sebuah aplikasi web yang dibangun menggunakan pustaka *React JS* sebagai *frontend library*-nya. *React* memiliki algoritma pembanding yang cerdas untuk memperbarui bagian-bagian tertentu dalam DOM-nya, sementara bagian lainnya tetap tidak terpengaruh [3]. *React* menggunakan virtual DOM yang dapat menentukan apakah komponen harus dimuat ulang atau tidak berdasarkan pada status komponen saat ini dan perubahan apa yang terjadi. Hal ini mencegah aplikasi dari proses merender ulang secara tidak perlu [2].

Sebagian besar aplikasi yang menggunakan *React*, akan mengaplikasikan proses *bundling*. *Bundling* merujuk pada proses menyusuri berkas-berkas yang diimpor dan menyatukannya menjadi satu berkas tunggal yang disebut "*bundle*". *Bundle* tersebut dimasukkan ke dalam halaman web untuk memuat seluruh aplikasi secara keseluruhan pada saat yang sama [5].

Namun seiring dengan pertumbuhan aplikasi, *bundle* juga akan mengalami peningkatan ukuran, terutama ketika menggunakan *library* pihak ketiga yang besar sehingga dapat mengakibatkan aplikasi memerlukan waktu yang lebih lama untuk dimuat. Pemecahan *bundle* dapat dilakukan untuk mengatasi permasalahan meningkatnya ukuran *bundle* [5].

Code Splitting

Code splitting adalah metode untuk meningkatkan kinerja aplikasi web dengan membagi kode JavaScript menjadi bagian-bagian kecil yang disebut *chunk*. *Chunk* kemudian dimuat ke *browser web* sesuai kebutuhan. Pemanfaatan *code splitting* dapat mengurangi waktu *initial render* aplikasi web dengan mengurangi jumlah JavaScript yang dikirimkan ke *browser*. Dengan kata lain, *code splitting* memungkinkan aplikasi untuk membagi kode JavaScript menjadi bagian-bagian yang lebih kecil, dan hanya memuat bagian yang diperlukan saat dibutuhkan. Hal ini mengurangi beban awal saat memuat aplikasi, sehingga mempercepat waktu muat halaman web tanpa harus mengirimkan keseluruhan kode *JavaScript* di awal [4].

React memiliki dukungan untuk memecah kode pada level komponen dengan menyediakan suatu fitur bawaan yang disebut *lazy* [5]. Fitur ini memungkinkan pemuatan dinamis komponen-komponen *React* ke dalam peramban. Melalui penggunaan *lazy*, aplikasi dapat menunda pemuatan komponen-komponen yang tidak digunakan hingga saat pengguna meminta akses ke komponen tersebut melalui interaksi dengan aplikasi. Salah satu pendekatan yang umum digunakan dalam memanfaatkan pemecahan kode pada aplikasi *React* adalah dengan memisahkan kode JavaScript untuk tampilan-tampilan yang berbeda ke dalam bagian-bagian terpisah. Pendekatan ini juga dikenal sebagai pemecahan kode pada level rute (*route-level code splitting*) [7].

Pada website ini, *code splitting* diimplementasikan pada setiap *route* yang terdiri dari sepuluh halaman. Penggunaan *React Lazy* digunakan untuk menangguhkan komponen terlebih dahulu sebelum di-*render* untuk pertama kali [8]. Pada saat ditangguhkan, terdapat komponen yang digunakan sebagai tampilan sementara menggunakan *React Suspense* [9].

Pengaplikasian *route* tersebut dibagi menjadi dua file, yaitu *public routes* dan *protected routes*. *Public routes* merupakan *routes* yang dapat diakses oleh pengguna tanpa melalui proses autentikasi, sedangkan *protected routes* merupakan *routes* yang hanya diakses oleh pengguna yang telah terautentikasi. Struktur folder *routing* pada *website* ini ditunjukkan pada Gambar 1.

```
Routes
|-- App.tsx
|-- AppRoutes.tsx
|-- ProtectedRoutes.tsx
|-- PublicRoutes.tsx
|-- RequireAuth.tsx
```

Gambar 1. Struktur Folder *Routing*

Code splitting diimplementasikan pada file ProtectedRoutes.tsx dan PublicRoutes.tsx. Pada kedua file tersebut, *React.Lazy* digunakan untuk membuat *lazy component*. *Lazy component* tersebut akan di-*render* hanya ketika komponen tersebut diperlukan. Berikut merupakan perbandingan kode program sebelum dilakukan *code splitting* (Tabel 1) dan setelah dilakukan *code splitting* (Tabel 2).

Tabel 1 Kode Program Sebelum dilakukan *Code Splitting*

```
1  import AddProjectPage from "@/pages/app/dosen/projects/add";
2  const protectedRoutes: IProps[] = [
3  {
4  path: "/app/dosen/projects/add",
5  element: <AddProjectPage />
6  },
7  ];
```

Tabel 2 Kode Program Setelah dilakukan *Code Splitting*

```
1  const LazyAddProjectPage = lazy(() => import("@/pages/app/dosen/projects/add"));
2  const protectedRoutes: IProps[] = [
3  {
4  path: "/app/dosen/projects/add",
5  element: <LazyAddProjectPage />
6  },
7  ];
```

Pada komponen *AppRoutes*, terdapat penggunaan *Suspense*. Komponen *Suspense* ini bertanggung jawab untuk menampilkan tampilan sementara saat proses *rendering* tampilan utama belum selesai. Tampilan sementara tersebut merupakan sesuatu yang dimasukkan ke dalam *props fallback*. Kode program pada Tabel 3 di merupakan penerapan *Suspense*.

Tabel 3 Implementasi *Routing* dan Penggunaan *Suspense*

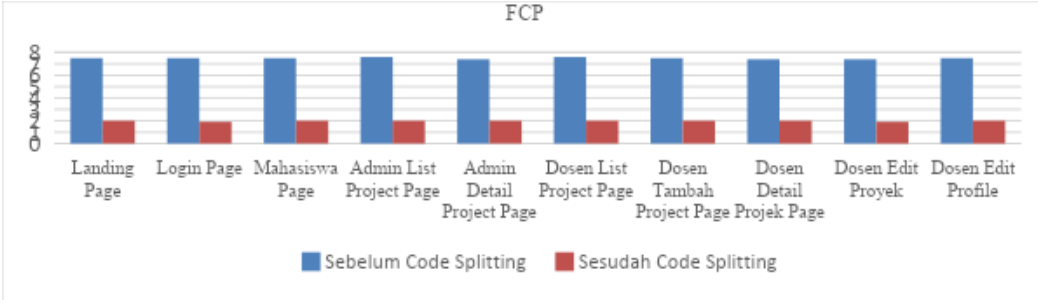
```
1  <Suspense fallback={<p>Loading...</p>}>
2  <Routes>
3  <Route element={<RequireAuth />}>
4  {protectedRoutes.map(({ path, element }) => (
5  <Route element={element} key={path} path={path} />
6  ))}
7  </Route>
8  </Routes>
9  </Suspense>
```

Selanjutnya, dilakukan pengukuran menggunakan Lighthouse yang memanfaatkan konsep *web vitals*. *Web vitals* diperkenalkan oleh Google pada tahun 2020 sebagai panduan untuk memastikan pengalaman pengguna yang lebih baik [10].

HASIL DAN PEMBAHASAN

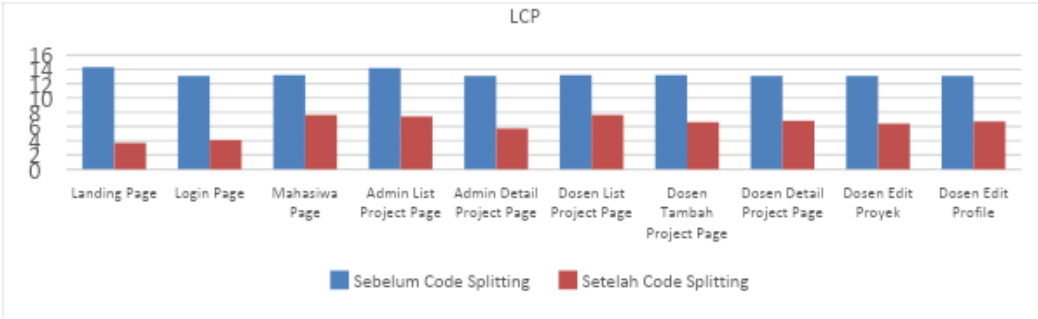
Lighthouse menggunakan beberapa parameter untuk mengukur performa suatu *website*, yaitu FCP (*First Contentful Paint*), LCP (*Largest Contentful Paint*), TBT (*Total Blocking Time*), CLS (*Cumulative Layout Shift*), dan *speed index*. Pengukuran dilakukan pada sebuah *website* yang sudah dioptimasi dan belum dioptimasi.

FCP merupakan salah satu dari parameter pengukur performa menggunakan Lighthouse. *First Contentful Paint* (FCP) menggambarkan interaktivitas dan merupakan periode antara input pertama pengguna dan respons halaman terhadap input tersebut sehingga semakin rendah waktu yang dibutuhkan maka performa *website* tersebut juga semakin baik. Gambar 2 merupakan perbandingan hasil pengukuran FCP sebelum dan sesudah dilakukan *code splitting*.



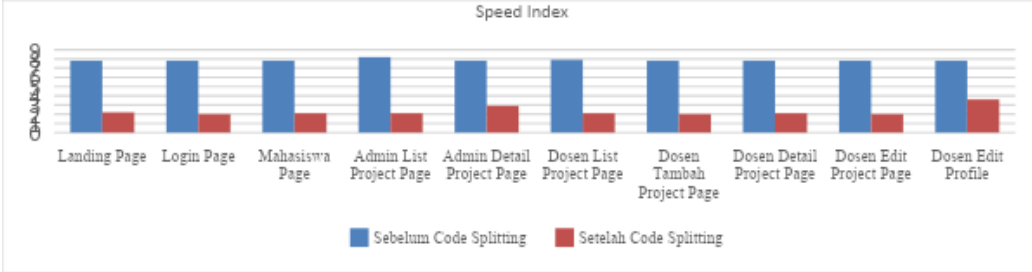
Gambar 2 Perbandingan Hasil pengukuran FCP Sebelum dan Sesudah dilakukan *Code Splitting*

Parameter selanjutnya adalah LCP atau *Largest Contentful Paint*. LCP mengukur waktu pemuatan elemen teks atau gambar terbesar yang terlihat di dalam *viewport* dalam satuan sekon. Semakin rendah waktu LCP maka performa *website* dinilai semakin baik. Hasil perbandingan pengukuran LCP sebelum dan sesudah dilakukan *code splitting* ditunjukkan pada Gambar 3.



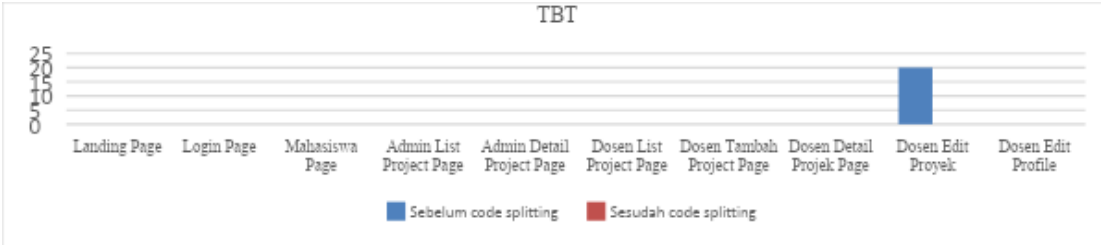
Gambar 3 Perbandingan Hasil Pengukuran LCP Sebelum dan Sesudah Dilakukan *Code Splitting*

Parameter pengukuran selanjutnya adalah *speed index* yang menunjukkan seberapa cepat konten ditampilkan secara visual selama pemuatan halaman. *Speed index* diukur menggunakan satuan sekon, yang mana semakin kecil angka semakin baik. Gambar 4 merupakan perbandingan hasil pengukuran *speed index* pada *website* sebelum dan sesudah diterapkan *code splitting*.



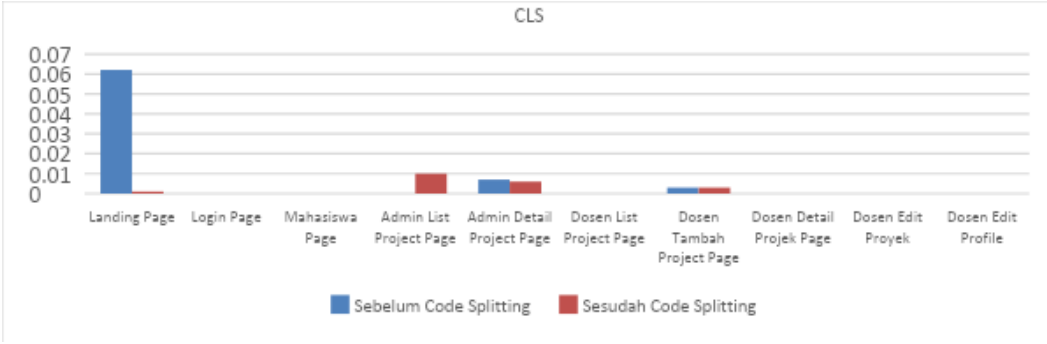
Gambar 4 Perbandingan Hasil Pengukuran *Speed Index* Sebelum dan Sesudah Dilakukan *Code Splitting*

Parameter selanjutnya adalah TBT atau *Total Blocking Time*. TBT mengukur berapa lama waktu halaman dalam kondisi melakukan block terhadap input pengguna, yaitu waktu antara *First Contentful Paint* dan *Time to Interactive*. TBT diukur menggunakan satuan milisekon dan semakin rendah nilainya semakin baik. Gambar 5 di merupakan hasil perbandingan pengukuran performa TBT dari suatu *website* sebelum dan setelah diterapkan *code splitting*.



Gambar 5 Perbandingan Hasil pengukuran TBT Sebelum dan Sesudah dilakukan *Code Splitting*

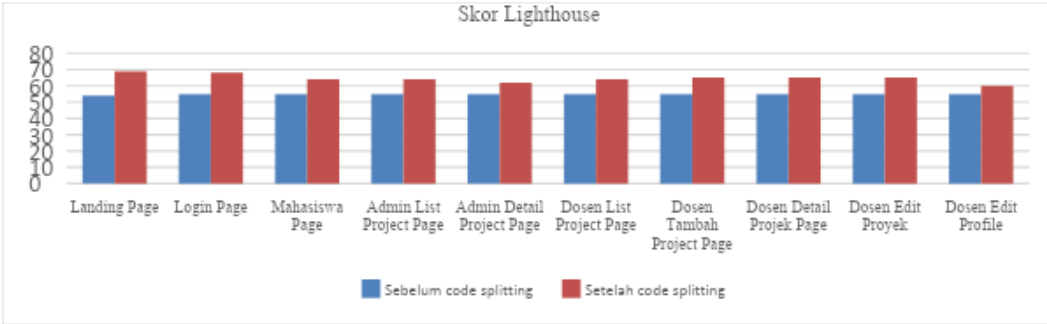
Selanjutnya, CLS atau *Cumulative Layout Shift* merupakan parameter dari uji performa menggunakan Lighthouse yang mengukur skor terhadap perubahan tata letak yang tidak terduga selama halaman aktif. Semakin rendah skor CLS, maka performa *website* dianggap semakin baik. Perbandingan pengukuran skor CLS suatu *website* sebelum dan sesudah diterapkan *code splitting*, ditunjukkan pada Gambar 6.



Gambar 6 Perbandingan Hasil pengukuran CLS Sebelum dan Sesudah Dilakukan *Code Splitting*

Parameter-parameter sebelumnya, diakumulasi oleh Lighthouse dengan perhitungan tertentu. Hasilnya, terdapat skor Lighthouse dengan rentang 1-100. Semakin tinggi skor tersebut,

maka performa *website* dinilai semakin baik. Berikut ini pada Gambar 7 merupakan perbandingan pengukuran skor Lighthouse pada suatu *website* sebelum dan setelah diterapkan *code splitting*.



Gambar 7 Perbandingan Hasil Pengukuran Skor Lighthouse Sebelum dan Sesudah Dilakukan *Code Splitting*

KESIMPULAN

Penelitian telah menunjukkan bahwa menerapkan teknik *code splitting* dapat secara signifikan meningkatkan performa sebuah *website*. Dalam pengujian yang menggunakan alat pengukuran performa seperti Lighthouse, hasilnya menunjukkan peningkatan yang konsisten dalam berbagai parameter kunci performa. Misalnya, waktu pemrosesan awal (FCP) dan waktu pemrosesan utama yang terbesar (LCP) dapat mengalami peningkatan yang signifikan setelah menerapkan *code splitting*. Begitu pula dengan waktu untuk berinteraksi pertama kali (TBT), stabilitas visual (CLS), indeks kecepatan (*speed index*), dan skor keseluruhan Lighthouse, semuanya menunjukkan perbaikan yang nyata. Perbaikan tersebut ditunjukkan dengan peningkatan rerata skor Lighthouse sebesar 54,9 yang meningkat menjadi 64,6.

Melalui pemisahan kode (*code splitting*), halaman web hanya memuat kode yang diperlukan untuk me-*render* bagian tertentu dari halaman tersebut. Ini memungkinkan *browser* untuk mengunduh dan mengeksekusi kode secara lebih efisien, mengurangi beban pada sumber daya jaringan dan meningkatkan responsifitas keseluruhan halaman. Dengan demikian, pengguna dapat mengalami waktu muat yang lebih cepat dan pengalaman pengguna yang lebih baik saat mengakses situs web yang dioptimalkan dengan baik.

Hasil ini menegaskan pentingnya penerapan praktik pengembangan yang disarankan seperti *code splitting* dalam upaya meningkatkan performa dan kualitas pengalaman pengguna pada *website*. Dengan menerapkan teknik ini, pengembang dapat mencapai standar performa yang lebih tinggi dan memberikan pengalaman yang lebih memuaskan bagi pengguna, yang pada gilirannya dapat berdampak positif pada tingkat retensi pengguna dan konversi di situs web.

DAFTAR PUSTAKA

[1] O. A. Abdurakhimovich, "Analyzing The Efficiency and Performance Optimization Techniques of React.Js in Modern Web Development," *ZDIT,* 2023.

[2] A. Javeed, "Performance Optimization Techniques for ReactJS," *IEEE International Conference on Electrical, Computer and Communication Technologies(ICECCT),* 2019.

[3] P. Rawat and A. N. Mahajan, "ReactJS: A Modern Web Development Framework," *International Journal of Innovative Science and Research Technology,* 2020.

[4] F. Pavić and L. Brkić , "Methods of Improving and Optimizing React Web Applications," *MIPRO,* 2021.

[5] React Development Team, "Code Splitting - React Documentation," [Online]. Terdapat: https://legacy.reactjs.org/docs/code-splitting.html. [Diakses pada 2023].

[6] D. Tanudjaja and R. Tanone, "Analisis Penerapan Code Splitting Library React pada Aplikasi Penjualan Mebel Berbasis Website," *Jurnal Teknik Informatika dan Sistem Informasi,* 2021.

[7] S. Lauria, "Improving the Initial Rendering Performance of React Applications Through Contemporary Rendering Approaches," *Otaniemi: Aalto University School of Science,* 2023.

[8] React Development Team, "React Reference - Lazy," [Online]. Terdapat: https://react.dev/reference/react/lazy. [Diakses pada 2024].

[9] React Development Team, "React Reference - Suspense," [Online]. Terdapat: https://react.dev/reference/react/Suspense. [Diakses pada 2024].

[10] N. Wehner, M. Amir, M. Seufert, R. Schatz and T. Hoßfeld, "A Vital Improvement? Relating Google's Core Web Vitals to Actual Web QoE," in *Quality of Multimedia Experience*, 2022.