BAB 1 PENDAHULUAN

1.1. Latar Belakang

Performa aplikasi web menjadi salah satu faktor utama dalam memberikan pengalaman pengguna yang optimal. Seiring dengan berkembangnya teknologi web, aplikasi modern semakin kompleks dengan fitur yang lebih banyak dan interaksi yang lebih dinamis. Hal ini berdampak pada peningkatan ukuran dan kompleksitas kode yang dijalankan di sisi klien. Oleh karena itu, tantangan yang dihadapi adalah bagaimana memastikan aplikasi tetap cepat, efisien, dan responsif, meskipun jumlah fitur terus bertambah[1].

React.js merupakan salah satu framework yang banyak digunakan dalam pengembangan aplikasi web karena efisiensi dalam rendering melalui Virtual DOM [2]. Namun, secara default, React menggunakan pendekatan single-bundle architecture, di mana seluruh kode aplikasi dikemas menjadi satu berkas JavaScript besar yang harus dimuat sebelum aplikasi dapat digunakan oleh pengguna. Arsitektur monolithic ini menciptakan performance debt [3]. Dengan bertambahnya fitur dan dependensi, ukuran bundle semakin besar, yang mengakibatkan peningkatan *initial loading time* serta penurunan responsivitas aplikasi [4].

Evaluasi awal yang dilakukan terhadap aplikasi menunjukkan bahwa ada masalah performa yang cukup signifikan, terutama pada performance score google lighthouse dan waktu muat awal halaman. Pengukuran awal menggunakan Google Lighthouse dengan mode desktop menggunakan simulated mode dengan device 1x CPU slowdown dan throughput network 10240 kilobits/second, mendapatkan hasil performa skor yang rendah, beberapa halaman memiliki Largest Contentful Paint (LCP) yang tinggi, Total Blocking Time (TBT) yang besar, serta Speed Index yang lambat. Nilai LCP yang tinggi menunjukkan bahwa elemen terbesar dalam halaman membutuhkan waktu lama untuk dimuat, sementara TBT yang besar menunjukkan bahwa skrip berat menghambat interaksi pengguna, Analisis awal menggunakan Webpack Bundle Analyzer mengungkapkan bahwa ukuran bundle aplikasi sebelum optimasi mencapai 15.28 MB (Stat Size), dengan Parsed Size sebesar 967.95 KB

dan Gzipped Size 304.33 KB. Rendahnya performance score juga terindikasi dengan rendahnya waktu muat awal yang disebabkan oleh keseluruhan bundle aplikasi yang diunduh pada saat pertama kali akses, hasil pengukuran waktu muat awal aplikasi berada di angka 3723ms atau 3.7 detik hampir 4 detik. Hasil ini sejalan dengan penelitian yang dilakukan oleh Filip Pavić dan Ljiljana Brkić yang menunjukkan bahwa peningkatan ukuran bundle menyebabkan penurunan skor performa pada Lighthouse dan *initial loading time* aplikasi [4], dan menurut penelitian dari Tanudjaja dan Tanone menyebutkan bahwa tingkat *rendering website* yang lambat dapat mempengaruhi *user experience*, dimana mereka menyimpulkan batas waktu muat yang wajar berada di 3 detik [5].

Untuk mengatasi permasalahan performa pada aplikasi React dengan pendekatan Client-Side Rendering (CSR), beberapa teknik optimasi dapat diterapkan. Teknik seperti *tree shaking* dan *minification* digunakan untuk mengurangi ukuran kode. *Tree shaking* bertujuan menghapus kode yang tidak digunakan dari bundle saat proses build, sedangkan minifiers menyederhanakan sintaksis kode dengan menghapus karakter yang tidak diperlukan seperti spasi dan komentar, serta memperpendek nama variabel. Namun, kedua teknik ini tidak memiliki dampak yang lebih luas dalam meningkatkan performa skor, memberikan pengurangan ukuran file yang terbatas dan tidak secara signifikan mengurangi waktu muat awal aplikasi karena kode utama tetap dimuat seluruhnya di awal [6].

Sebagai alternatif yang lebih adaptif, penelitian ini berfokus pada optimasi menggunakan teknik *code splitting* dan *lazy loading*. Code splitting memungkinkan pemisahan kode aplikasi menjadi beberapa bagian lebih kecil (modular bundle) yang dapat dimuat hanya ketika dibutuhkan, sementara *lazy loading* akan menunda pemuatan komponen atau aset hingga pengguna benar-benar membutuhkannya.

Pendekatan ini dipilih karena memiliki dampak yang lebih luas terhadap peningkatan skor performa aplikasi, pengurangan ukuran bundle awal, dan peningkatan efisiensi rendering. Turcotte et al. menyatakan bahwa dibandingkan dengan *minification* dan *tree shaking*, strategi berbasis *code splitting* dan *lazy loading* memberikan hasil yang lebih optimal dalam skenario client-side, karena mempertimbangkan beban kode aktual yang diperlukan pada setiap tahapan

penggunaan aplikasi [6]. Dengan demikian, penerapan *code splitting* dan *lazy loading* menjadi solusi yang lebih adaptif dan scalable dalam mengoptimalkan performa aplikasi client-side.

Dengan menerapkan teknik *code splitting* dan *lazy loading*, aplikasi diharapkan mengalami pengurangan ukuran bundle awal yang signifikan, mempercepat waktu muat awal, serta meningkatkan skor performa pada Google Lighthouse. Penelitian oleh Malavolta et al. mendukung hal ini dengan menunjukkan bahwa implementasi sistematis dari teknik tersebut, mampu menurunkan ukuran bundle lebih dari 80% dan mempercepat waktu muat hingga 80.66% [7]. Pengujian optimasi akan dilakukan menggunakan metrik seperti Loading Time, Bundle Size, Performance Score, First Contentful Paint (FCP), Largest Contentful Paint (LCP), Total Blocking Time (TBT), Cumulative Layout Shift, dan Speed Index, untuk mengevaluasi dampak dari penerapan teknik ini terhadap peningkatan performa dan kecepatan pemuatan aplikasi React.

1.2. Rumusan Masalah

Penelitian ini berfokus pada optimasi performa aplikasi web berbasis React dengan menerapkan teknik *code splitting* dan *lazy loading*. Berdasarkan latar belakang yang telah diuraikan, penelitian ini merumuskan beberapa pertanyaan utama :

1. Bagaimana dampak penerapan teknik *Code Splitting* dan *Lazy Loading* terhadap *Initial Loading Time* dan *Initial Bundle Size* pada aplikasi React?
2. Bagaimana Perubahan performa aplikasi berdasarkan metrik evaluasi dari Google Lighthouse setelah optimasi dilakukan?

1.3. Tujuan dan Manfaat

Penelitian ini bertujuan untuk menganalisis efektivitas teknik *code splitting* dan *lazy loading* dalam meningkatkan performa aplikasi web berbasis React. Secara spesifik, penelitian ini memiliki tujuan sebagai berikut:

1. Mengimplementasikan dan mengevaluasi dampak penerapan *Code Splitting* dan *Lazy Loading* terhadap ukuran bundle dan waktu muat awal aplikasi.
2. Menganalisis peningkatan performa berdasarkan metrik evaluasi seperti LCP, FCP, TBT, CLS dan Speed Index setelah optimasi dilakukan.

1.4. Batasan Masalah

Penelitian ini memiliki beberapa batasan-batasan supaya penelitian bisa berjalan dengan efisien dan mempunyai ruang untuk perbaikan. Adapun batasan masalah dalam penelitian ini adalah sebagai berikut :

1. Ruang Lingkup Aplikasi:
    a. Penelitian berfokus pada pengujian aplikasi web manajemen restoran Antria.
    b. Menggunakan versi React terbaru.
2. Parameter pengukuran:
    a. Waktu muat (*Loading Time*) aplikasi.
    b. Ukuran file komponen dimuat pertama kali (*Initial Bundle Size*)
    c. Metrik performa skor Google Lighthouse.
3. Lingkungan Pengembangan dan pengujian :
    a. Pengujian dilakukan pada satu perangkat penelitian.
    b. Pengukuran metrik Lighthouse dilakukan pada simulated device mode Lighthouse dengan device 1x CPU slowdown dan throughput network 10240 kilobits/second.
4. Cakupan Optimasi: Penelitian ini tidak mencakup optimasi aset visual karena pengujian awal dilakukan dengan aset beresolusi tinggi untuk menjaga konsistensi perbandingan hasil utama sebelum dan sesudah penerapan *code splitting* dan *lazy loading*.

1.5. Metode Penelitian

Penelitian ini dilakukan dengan pendekatan studi literatur, pengukuran empiris, analisis statistik, simulasi, perancangan, dan implementasi. Langkah-langkah metodologi penelitian ini terdiri dari beberapa tahapan sebagai berikut:

1. **Studi Literatur**

    Studi literatur dilakukan dengan memperoleh pemahaman mendalam mengenai performa aplikasi web, khususnya melalui teknik *code splitting* dan *lazy loading* dalam React. Referensi yang dikaji mencakup:

    - Penelitian terdahulu mengenai optimasi performa aplikasi web dengan *code splitting* dan *lazy loading*.

- Dokumentasi resmi React serta *best practices* terkait optimasi performa aplikasi berbasis React.

2. **Pengukuran Empiris**

   Untuk menilai performa aplikasi sebelum dan sesudah optimasi, dilakukan pengukuran empiris melalui beberapa metrik utama:

   - Waktu muat halaman (Page Load Time) menggunakan Chrome DevTools.
   - Ukuran bundel aplikasi dengan webpack-bundle-analyzer untuk mengevaluasi distribusi dan pengelompokan modul.
   - Metrik performa skor menggunakan Google Lighthouse, yang mencakup Performance score, First Contentful Paint (FCP), Largest Contentful Paint (LCP), Speed Index, Total Blocking Time dan Cumulative Layout Shift (CLS).

3. **Perancangan dan Implementasi**

   Penelitian ini menggunakan pendekatan eksperimental dengan menerapkan teknik optimasi performa melalui beberapa tahap:

   1. Persiapan
      a. Memahami karakteristik aplikasi yang akan dioptimasi.
      b. Menentukan metrik performa yang akan menjadi fokus evaluasi.
      c. Mengonfigurasi lingkungan pengujian untuk memastikan hasil yang konsisten.
   2. Pengukuran performa awal sebelum optimasi
   3. Implementasi
      a. Menerapkan *Code Splitting* untuk memisahkan modul aplikasi menjadi bundle yang lebih kecil.
      b. Menerapkan *Lazy Loading* untuk memuat komponen secara dinamis sesuai kebutuhan pengguna.
   4. Pengujian pengukuran performa setelah implementasi optimasi.

4. **Analisis dan Evaluasi**

Setelah implementasi, hasil penelitian dianalisis secara kuantitatif dengan membandingkan metrik performa sebelum dan sesudah optimasi serta penyusunan rekomendasi berdasarkan penelitian.

5. **Dokumentasi dan Penyusunan Laporan**

5.1. Jadwal Pelaksanaan

Berikut ini adalah jadwal pelaksanaan dari penelitian tugas akhir ini:

Tabel 1.1. Jadwal Pelaksanaan Tugas Akhir

No.	Deskripsi Tahapan	Bulan 1	Bulan 2	Bulan 3	Bulan 4	Bulan 5	Bulan 6
1	Studi Literatur						
2	Analisa dan Pengumpulan Data						
3	Perancangan dan Implementasi						
4	Pengujian dan Analisis						
5	Evaluasi						
6	Penyusunan Laporan/Buku TA						