PENGEMBANGAN WEB (TEORI)

LAPORAN EKSPERIMEN MENGENAI PERBANDINGAN MICRO FRONTEND VS MONOLITH FRONTEND DITINJAU DARI PERFORMANCE

Laporan ini disusun untuk memenuhi tugas 1 mata kuliah Pengembangan Web (Teori)

Disusun oleh kelompok B4:

Asri Husnul Rosadi	221524035
Faris Abulkhoir	221524040
Mahardika Pratama	221524044
Muhamad Fahri Yuwan	221524047
Najib Alimudin Fajri	221524053
Septyana Agustina	221524058
Sarah	221524059

Dosen Pengampu:
Joe Lian Min, M.Eng.

JURUSAN TEKNIK KOMPUTER DAN INFORMATIKA
PROGRAM STUDI D4 TEKNIK INFORMATIKA
POLITEKNIK NEGERI BANDUNG

2024

DAFTAR ISI

A. IDENTIFIKASI PROBLEM

Ketika membandingkan arsitektur Monolith Frontend dengan Micro Frontend, penting untuk mengevaluasi bagaimana masing-masing pendekatan mempengaruhi performa aplikasi web. Performansi aplikasi web menjadi faktor krusial yang mempengaruhi pengalaman pengguna, kecepatan muat, dan responsivitas aplikasi. Aplikasi yang lebih cepat dan responsif tidak hanya memberikan pengalaman pengguna yang lebih baik tetapi juga dapat meningkatkan efisiensi pengembangan dan pemeliharaan. Tujuan utama dari eksperimen ini adalah untuk menilai perbedaan performa antara aplikasi berbasis Monolith Frontend dan Micro Frontend dengan mengukur metrik performa yang relevan.

B. DESKRIPSI PROBLEM

Masalah yang dihadapi adalah bagaimana arsitektur Monolith Frontend dan Micro Frontend mempengaruhi performa aplikasi web. Monolith Frontend mengintegrasikan semua komponen dan logika aplikasi dalam satu basis kode, yang seringkali menghasilkan ukuran bundel yang besar dan waktu muat yang lebih lama. Di sisi lain, Micro Frontend membagi aplikasi menjadi bagian-bagian independen, yang dapat meningkatkan skalabilitas dan memungkinkan pengelolaan yang lebih fleksibel. Namun, perbedaan dalam arsitektur ini dapat mempengaruhi metrik performa seperti waktu muat, waktu interaksi, dan stabilitas layout, yang memerlukan evaluasi mendalam untuk menentukan pendekatan yang lebih efisien.

C. METODOLOGI EKSPERIMEN

1) Tujuan

   Tujuan dari eksperimen ini adalah untuk membandingkan performansi aplikasi web yang dikembangkan dengan arsitektur Monolith Frontend dan Micro Frontend. Evaluasi dilakukan dengan mengukur berbagai metrik performansi untuk menilai kecepatan muat, responsivitas, dan stabilitas visual masing-masing pendekatan.

2) Desain Eksperimen

   - Setup Environment
     - Lingkungan Pengujian: Gunakan lingkungan pengujian yang sama untuk kedua arsitektur untuk memastikan hasil yang konsisten. Jalankan aplikasi di server lokal atau lingkungan staging yang terisolasi.

- Alat Pengukuran: Gunakan Google Lighthouse untuk mengukur metrik performansi. Lighthouse adalah alat yang menyediakan laporan terperinci tentang waktu muat, interaksi pengguna, dan stabilitas layout.

3) Variabel Eksperimen
   - Variabel Bebas:
     - Arsitektur Aplikasi: Perbedaan antara Monolith Frontend dan Micro Frontend. Ini adalah variabel utama yang mempengaruhi hasil eksperimen.
   - Variabel Terikat:
     - First Contentful Paint (FCP): Waktu yang diperlukan untuk menampilkan elemen pertama di layar.
     - Largest Contentful Paint (LCP): Waktu yang diperlukan untuk menampilkan elemen terbesar di layar.
     - Total Blocking Time (TBT): Waktu total di mana thread utama diblokir, mempengaruhi interaksi pengguna.
     - Cumulative Layout Shift (CLS): Perubahan layout yang mempengaruhi pengalaman visual.
     - Speed Index**:** Kecepatan di mana konten halaman menjadi terlihat.
4) Langkah-Langkah Pengujian
   - Pengembangan Aplikasi:
     - Monolith Frontend:
       - Inisialisasi Proyek: Buat aplikasi React dengan Create React App.
       - Pengembangan: Kembangkan aplikasi dengan semua komponen terintegrasi dalam satu basis kode.
     - Micro Frontend:
       - Inisialisasi Proyek: Buat tiga proyek React terpisah untuk host-app, header-microfrontend, dan maincontent-microfrontend.

        - Pengembangan: Implementasikan micro frontends di masing-masing proyek dan integrasikan dalam host-app.

- Pengukuran Performansi:
    - Monolith Frontend:
        - Jalankan aplikasi dan gunakan Google Lighthouse untuk mengukur metrik performansi.
    - Micro Frontend:
        - Jalankan aplikasi host-app yang mengintegrasikan micro frontends dan ukur metrik performansi menggunakan Google Lighthouse.

5) Pengumpulan Data:
    - Catat hasil metrik performansi dari Google Lighthouse untuk kedua arsitektur. Hasil ini mencakup First Contentful Paint, Largest Contentful Paint, Total Blocking Time, Cumulative Layout Shift, dan Speed Index.
6) Analisis Data:
    - Bandingkan hasil metrik performansi antara Monolith Frontend dan Micro Frontend untuk menilai perbedaan performa.
7) Pertimbangan dan Variabel Kontrol
    - Kondisi Pengujian: Pastikan kondisi pengujian konsisten, termasuk kecepatan internet, perangkat keras, dan perangkat lunak yang digunakan.
    - Versi Browser: Gunakan versi browser yang sama untuk memastikan hasil pengukuran yang akurat.

D. PELAKSANAAN EKSPERIMEN

1) Setup dan Konfigurasi:

- Setup Environment**:**
    - Lingkungan Pengujian: Gunakan server lokal untuk meng-host aplikasi. Pastikan tidak ada aplikasi lain yang berjalan yang dapat mempengaruhi hasil pengujian.
    - Alat Pengukuran: Instal Google Lighthouse sebagai ekstensi di browser Chrome atau gunakan alat lain seperti WebPageTest untuk mengukur metrik performansi.

2) Pengembangan Aplikasi
   - Monolith Frontend:
     - Inisialisasi Proyek:
       - Jalankan perintah berikut untuk membuat aplikasi React baru:

```
npx create-react-app monolith-frontend
cd monolith-frontend
```

     - Pengembangan
       - src/index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import reportWebVitals from
'./reportWebVitals';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);

// Mengukur performa aplikasi
reportWebVitals(console.log);
```

       - src/App.js

```
import React, { useState, useEffect } from
'react';
import './App.css';

const App = () => {
  const [data, setData] = useState(null);

  useEffect(() => {
    // Simulasi pengambilan data dari API
    const fetchData = async () => {
      const response = await
fetch('https://jsonplaceholder.typicode.com/pos
ts');
      const result = await response.json();
      setData(result);
    };

    fetchData();
  }, []);

  return (
    <div className="App">
      <header className="App-header">
```

```
        <h1>Monolith Frontend</h1>
        {data ? (
          <ul>
            {data.slice(0, 5).map(post => (
              <li
key={post.id}>{post.title}</li>
            ))}
          </ul>
        ) : (
          <p>Loading...</p>
        )}
      </header>
    </div>
  );
};

export default App;
```

- src/App.css

```
.App {
  text-align: center;
}

.App-logo {
  height: 40vmin;
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  color: white;
}

.App-link {
  color: #61dafb;
}
```

- Micro Frontend
  - Inisialisasi Proyek:
    - Jalankan perintah berikut untuk membuat aplikasi React baru:

```
npx create-react-app host-app
npx create-react-app header-microfrontend
npx create-react-app maincontent-microfrontend
```

  - Install dependencies

```
npm install webpack webpack-cli html-webpack-plugin
webpack-dev-server --save-dev
```

- Pengembangan
  - Header Micro Frontend
    header-microfrontend/webpack.config.js

```
const ModuleFederationPlugin =
require('webpack/lib/container/ModuleFederation
Plugin');
const packageJson = require('./package.json');

module.exports = {
  mode: 'development',
  devServer: {
    port: 3001,
  },
  plugins: [
    new ModuleFederationPlugin({
      name: 'headerMicrofrontend',
      filename: 'remoteEntry.js',
      exposes: {
        './Header': './src/Header',
      },
      shared: packageJson.dependencies,
    }),
  ],
  resolve: {
    extensions: ['.js', '.jsx'],
  },
};
```

header-microfrontend/src/Header.js

```
import React from 'react';

const Header = () => {
  return (
    <header style={{ padding: '20px',
backgroundColor: 'lightblue' }}>
      <h1>Micro Frontend Header</h1>
    </header>
  );
};

export default Header;
```

header-microfrontend/src/index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import Header from './Header';
```

```
ReactDOM.render(
  <React.StrictMode>
    <Header />
  </React.StrictMode>,
  document.getElementById('root')
);
```

- content-microfrontend
  content-microfrontend/webpack.config.js

```
const ModuleFederationPlugin =
require('webpack/lib/container/ModuleFederation
Plugin');
const packageJson = require('./package.json');

module.exports = {
  mode: 'development',
  devServer: {
    port: 3002,
  },
  plugins: [
    new ModuleFederationPlugin({
      name: 'contentMicrofrontend',
      filename: 'remoteEntry.js',
      exposes: {
        './Content': './src/Content',
      },
      shared: packageJson.dependencies,
    }),
  ],
  resolve: {
    extensions: ['.js', '.jsx'],
  },
};
```

content-microfrontend/src/Content.js

```
import React from 'react';

const Content = () => {
  return (
    <main style={{ padding: '20px',
backgroundColor: 'lightgreen' }}>
      <h2>Micro Frontend Content</h2>
      <p>This is the content of the micro
frontend.</p>
    </main>
  );
};

export default Content;
```

content-microfrontend/src/index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import Content from './Content';

ReactDOM.render(
  <React.StrictMode>
    <Content />
  </React.StrictMode>,
  document.getElementById('root')
);
```

- host-app
  host-app/webpack.config.js

```
const ModuleFederationPlugin =
require('webpack/lib/container/ModuleFederation
Plugin');
const packageJson = require('./package.json');

module.exports = {
  mode: 'development',
  devServer: {
    port: 3000,
  },
  plugins: [
    new ModuleFederationPlugin({
      name: 'hostApp',
      remotes: {
        Header:
'headerMicrofrontend@http://localhost:3001/remo
teEntry.js',
        Content:
'contentMicrofrontend@http://localhost:3002/rem
oteEntry.js',
      },
      shared: packageJson.dependencies,
    }),
  ],
  resolve: {
    extensions: ['.js', '.jsx'],
  },
};
```

host-app/src/App.js

```
import React from 'react';
import dynamic from 'react-dynamic-import';

const Header = dynamic(() =>
import('Header/Header'));
const Content = dynamic(() =>
import('Content/Content'));

const App = () => {
```

```
  return (
    <div>
      <Header />
      <Content />
    </div>
  );
};

export default App;
```

host-app/src/index.js

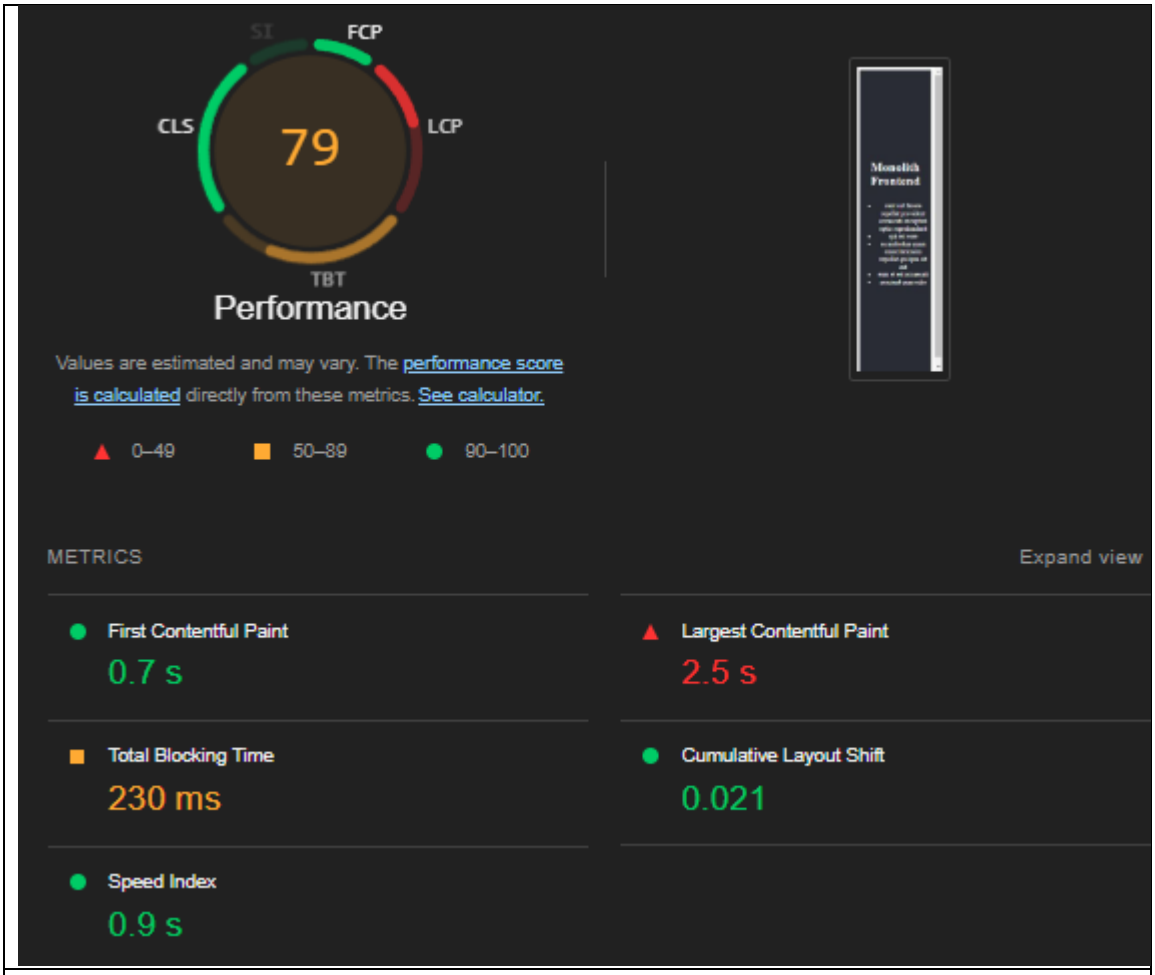
```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

- Jalankan aplikasi

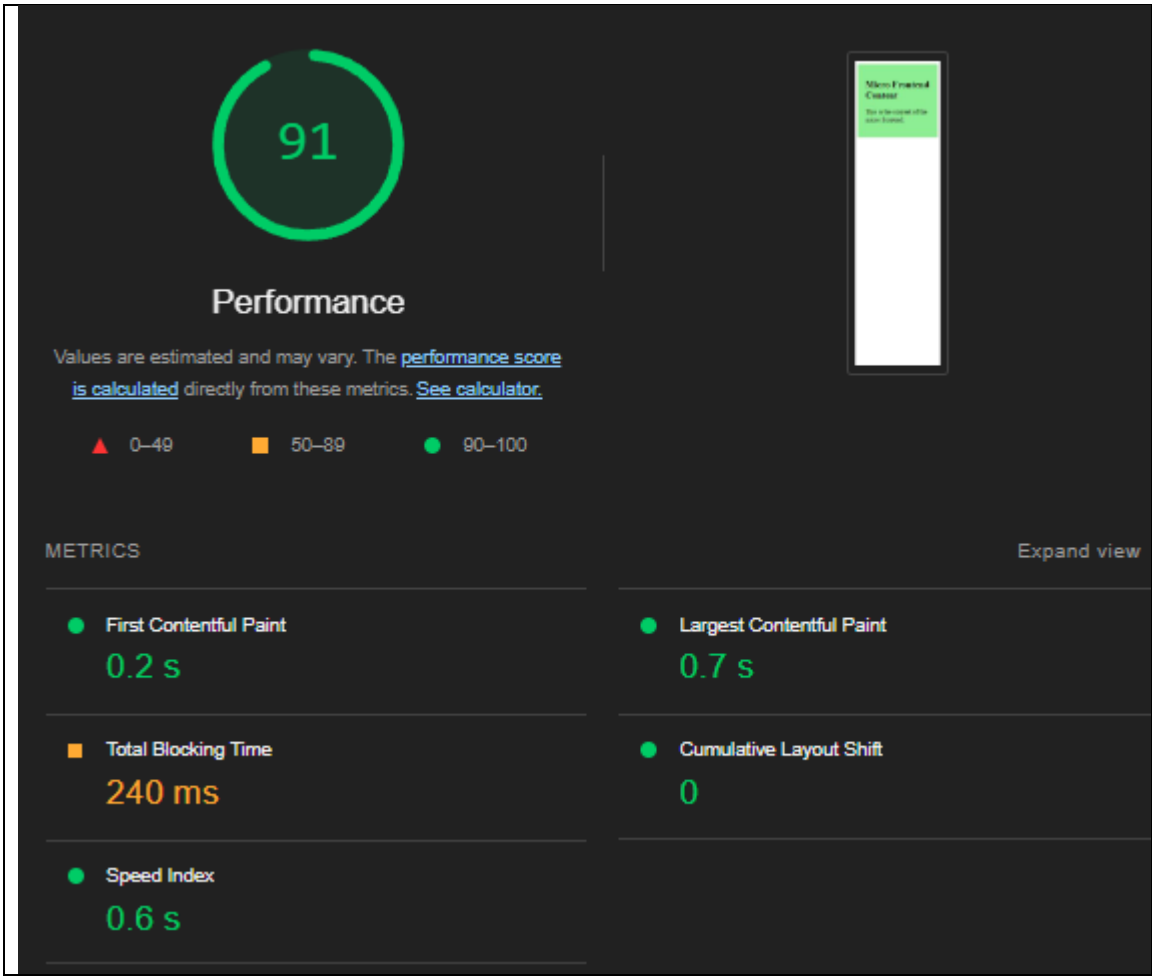
```
npm start --port 3001   # Untuk header-
microfrontend
npm start --port 3002   # Untuk maincontent-
microfrontend
npm start --port 3000   # Untuk host-app
```

3) Hasil Performance

Monolith Frontend



Micro Frontend



E. ANALISIS HASIL EKSPERIMEN

Aplikasi Monolith Frontend memiliki skor performa 79 dengan metrik waktu muat yang lebih lambat dibandingkan dengan Micro Frontend. First Contentful Paint dan Largest Contentful Paint menunjukkan waktu muat yang lebih lama, serta Total Blocking Time yang lebih tinggi, yang mempengaruhi interaksi pengguna. Cumulative Layout Shift menunjukkan sedikit pergeseran layout, yang dapat mempengaruhi pengalaman visual.

Aplikasi Micro Frontend menunjukkan performa superior dengan skor 91. Metrik performa seperti First Contentful Paint dan Largest Contentful Paint menunjukkan waktu muat yang jauh lebih cepat. Total Blocking Time yang lebih rendah dan Cumulative Layout Shift yang nol menunjukkan bahwa aplikasi lebih responsif dan stabil secara visual. Speed Index yang lebih rendah menunjukkan bahwa konten halaman menjadi terlihat lebih cepat.

F. KESIMPULAN

Micro Frontend menunjukkan performa yang lebih baik dibandingkan dengan Monolith Frontend, dengan skor performansi 97 versus 79. Penggunaan arsitektur Micro Frontend

memberikan keuntungan signifikan dalam waktu muat, responsivitas, dan stabilitas layout, menjadikannya pilihan yang lebih efisien untuk aplikasi dengan kebutuhan performa tinggi dan skalabilitas. Sementara itu, Monolith Frontend mungkin lebih mudah diimplementasikan dan dikelola untuk aplikasi yang lebih sederhana, namun dapat menghadapi masalah performa seiring dengan peningkatan kompleksitas aplikasi.