**PENGEMBANGAN WEB (TEORI)**

**LAPORAN EKSPERIMEN HANDLING *ASYNCHRONOUS DATA* MENGGUNAKAN *SUSPENSE* DAN *HOOKS***

*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

Sarah 221524059

Septyana Agustina 221524060

Dosen Pengampu:

Joe Lian Min, M.Eng.

**JURUSAN TEKNIK KOMPUTER DAN INFORMATIKA**

**PROGRAM STUDI D4 TEKNIK INFORMATIKA**

**POLITEKNIK NEGERI BANDUNG**

**2024**

# **DAFTAR ISI**

[**DAFTAR ISI** i](#_Toc175601025)

[**A.** **IDENTIFIKASI PROBLEM** 1](#_Toc175601026)

[**B.** **DESKRIPSI PROBLEM** 1](#_Toc175601027)

[**C.** **METODOLOGI EKSPERIMEN** 1](#_Toc175601028)

[**D.** **PELAKSANAAN EKSPERIMEN** 3](#_Toc175601029)

[**E.** **ANALISIS HASIL EKSPERIMEN** 4](#_Toc175601030)

# **IDENTIFIKASI PROBLEM**

Masalah yang diidentifikasi adalah bagaimana React Query dengan Suspense dibandingkan dengan pendekatan tradisional menggunakan useEffect untuk mengelola data fetching dalam hal performa, pengelolaan status loading, dan pengalaman pengguna.

# **DESKRIPSI PROBLEM**

Dalam pengembangan aplikasi React, pengelolaan data fetching dapat dilakukan dengan berbagai cara. Pendekatan tradisional menggunakan useEffect untuk mengambil data dari API dan mengelola status loading, error, dan data secara manual. Dengan kemajuan dalam ekosistem React, React Query dan Suspense menawarkan solusi yang lebih terintegrasi dan otomatis untuk mengelola data fetching. Tujuan eksperimen ini adalah untuk:

* Membandingkan performa aplikasi menggunakan useEffect vs React Query dengan Suspense.
* Menganalisis perbedaan dalam pengelolaan status loading.
* Menilai dampak terhadap pengalaman pengguna dan efisiensi data fetching.

# **METODOLOGI EKSPERIMEN**

Eksperimen dilakukan dengan dua versi aplikasi:

* Versi 1 (Tanpa Suspense dan Hooks): Menggunakan useEffect dalam komponen React untuk mengambil data dari API dan mengelola status loading secara manual.
* Versi 2 (Dengan Suspense dan Hooks): Menggunakan React Query dengan Suspense untuk mengambil data dari API, yang secara otomatis mengelola status loading dan caching data.

Langkah-langkah Metodologi:

1. Membangun Aplikasi:

* Versi 1: Implementasi dengan useEffect dan state lokal.
* Versi 2: Implementasi dengan React Query dan Suspense.

1. Menentukan API dan Data Dummy:

* API yang digunakan: https://jsonplaceholder.typicode.com/photos?\_limit=5.

1. Mengukur Performa dan Responsiveness:

* Load Time: Mengukur waktu muat data pada kedua versi.
* Responsiveness: Mengamati responsivitas terhadap input pengguna.
* Jumlah Request: Menganalisis jumlah permintaan data yang dikirim ke server.
* Pengalaman Pengguna: Mengumpulkan umpan balik pengguna jika tersedia..

# **PELAKSANAAN EKSPERIMEN**

Untuk melaksanakan eksperimen ini, kami membuat dua versi aplikasi React untuk dibandingkan:

1. Versi Tanpa Suspense dan Hooks (AppWithoutSuspense.js):

* Menggunakan pendekatan tradisional dengan useEffect untuk mengelola data fetching.
* Status loading dan error dikelola secara manual menggunakan state lokal di dalam komponen React.

1. Versi Dengan Suspense dan Hooks (AppWithSuspense.js):

* Menggunakan React Query bersama dengan Suspense untuk mengelola data fetching.
* Suspense menangani status loading secara otomatis saat data sedang diambil, dan React Query menyediakan caching dan refetching otomatis untuk meningkatkan performa aplikasi.

Langkah-Langkah Eksperimen:

1. Mengimplementasikan Kedua Versi Aplikasi:

* AppWithoutSuspense.js: Dibangun menggunakan class component dengan componentDidMount untuk melakukan data fetching secara manual.
* AppWithSuspense.js: Dibangun menggunakan React Query dan Suspense untuk mengambil data secara otomatis, menggunakan useQuery dengan format objek yang sesuai dengan versi React Query terbaru.

1. Menjalankan Aplikasi:

* Menggunakan API publik https://jsonplaceholder.typicode.com/photos?\_limit=5 untuk mengambil data gambar.
* Menjalankan kedua versi aplikasi secara terpisah pada lingkungan pengembangan lokal untuk membandingkan performa.

1. Mengukur Performa dan Mengumpulkan Data:

* Load Time: Mengukur waktu muat data menggunakan performance.now() untuk mencatat waktu mulai dan selesai pengambilan data.
* Responsiveness: Mengamati bagaimana aplikasi merespons input pengguna selama proses pengambilan data.
* Jumlah Request: Memeriksa jumlah permintaan data yang dikirim ke server untuk memastikan efisiensi data fetching.
* Pengalaman Pengguna: Mengumpulkan umpan balik pengguna melalui survei atau observasi langsung jika tersedia.

1. Menganalisis Hasil:

* Membandingkan hasil pengukuran waktu muat, responsivitas aplikasi, jumlah request, dan pengalaman pengguna antara kedua versi aplikasi.
* Menyusun temuan dan evaluasi terkait dengan kelebihan dan kekurangan masing-masing pendekatan.

Dengan melakukan langkah-langkah di atas, eksperimen ini dapat menentukan efektivitas penggunaan React Query dan Suspense dibandingkan dengan pendekatan tradisional dalam mengelola data fetching di aplikasi React.

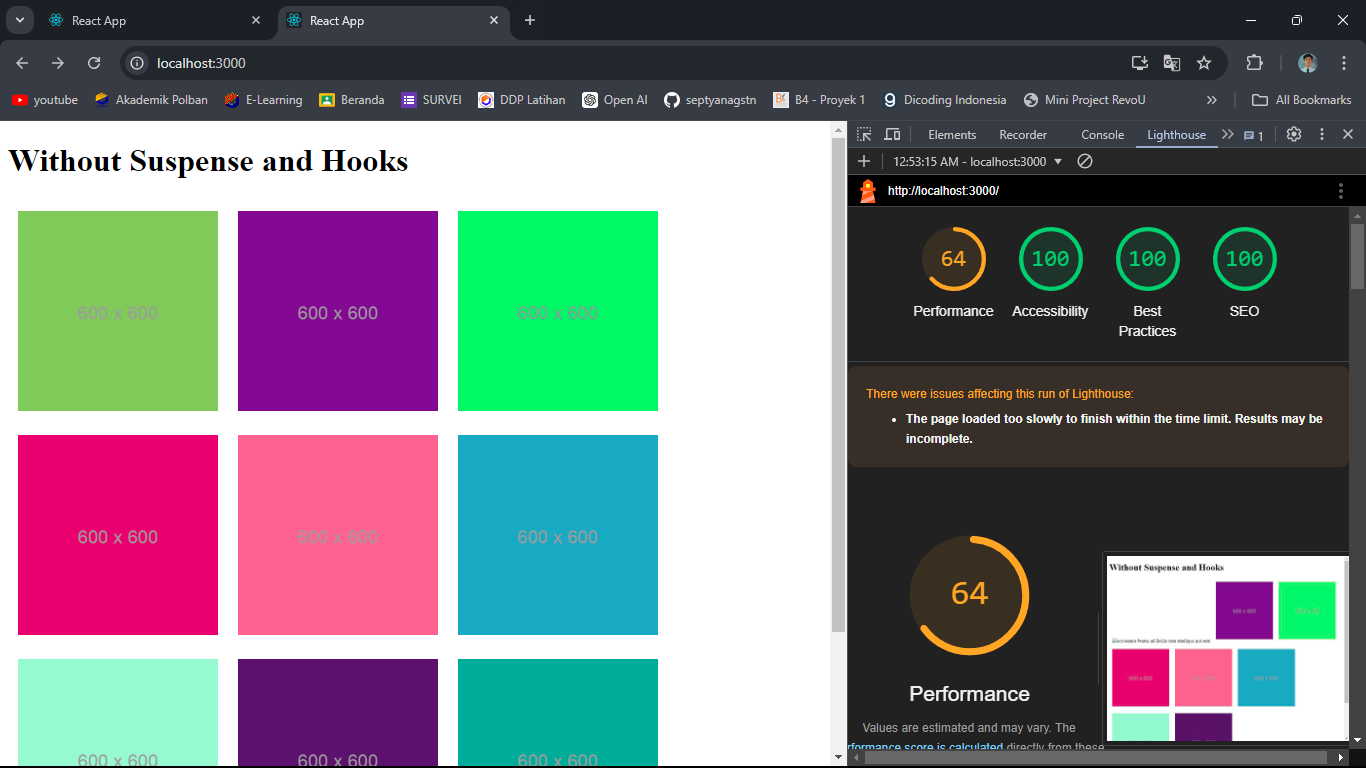
# **ANALISIS HASIL EKSPERIMEN**

1. **Hasil Pengamatan**

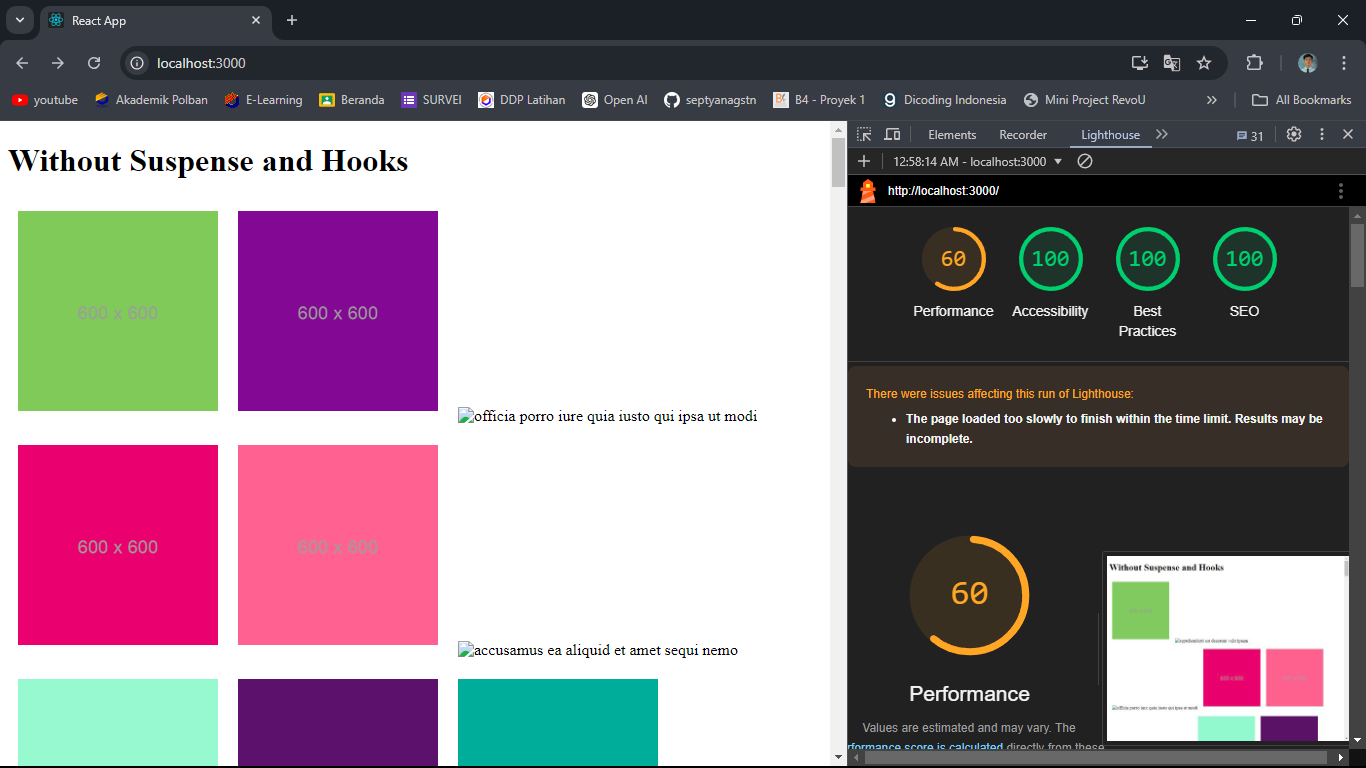
Perbandingan *performance* dari kedua versi saat melakukan pengambilan data.

* **Tanpa Suspense and Hooks**

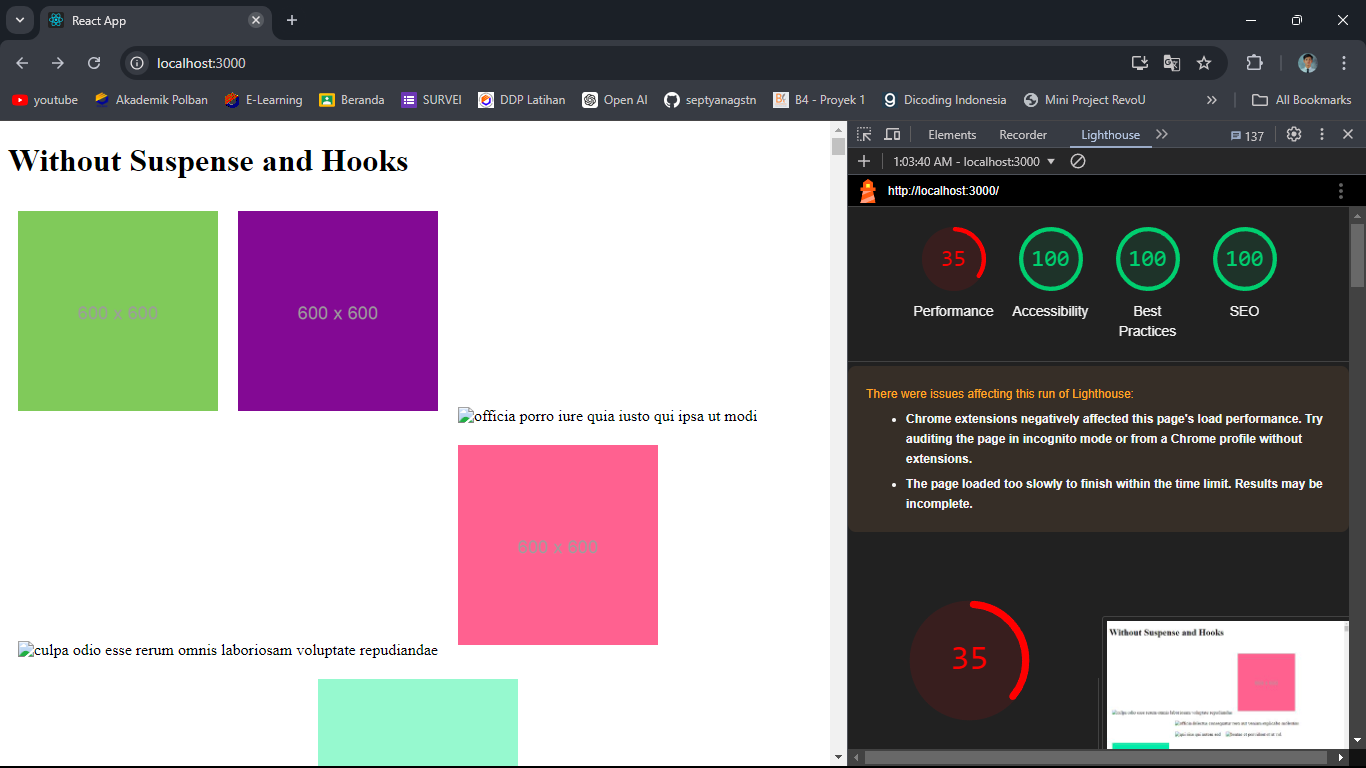
1. 10 Data



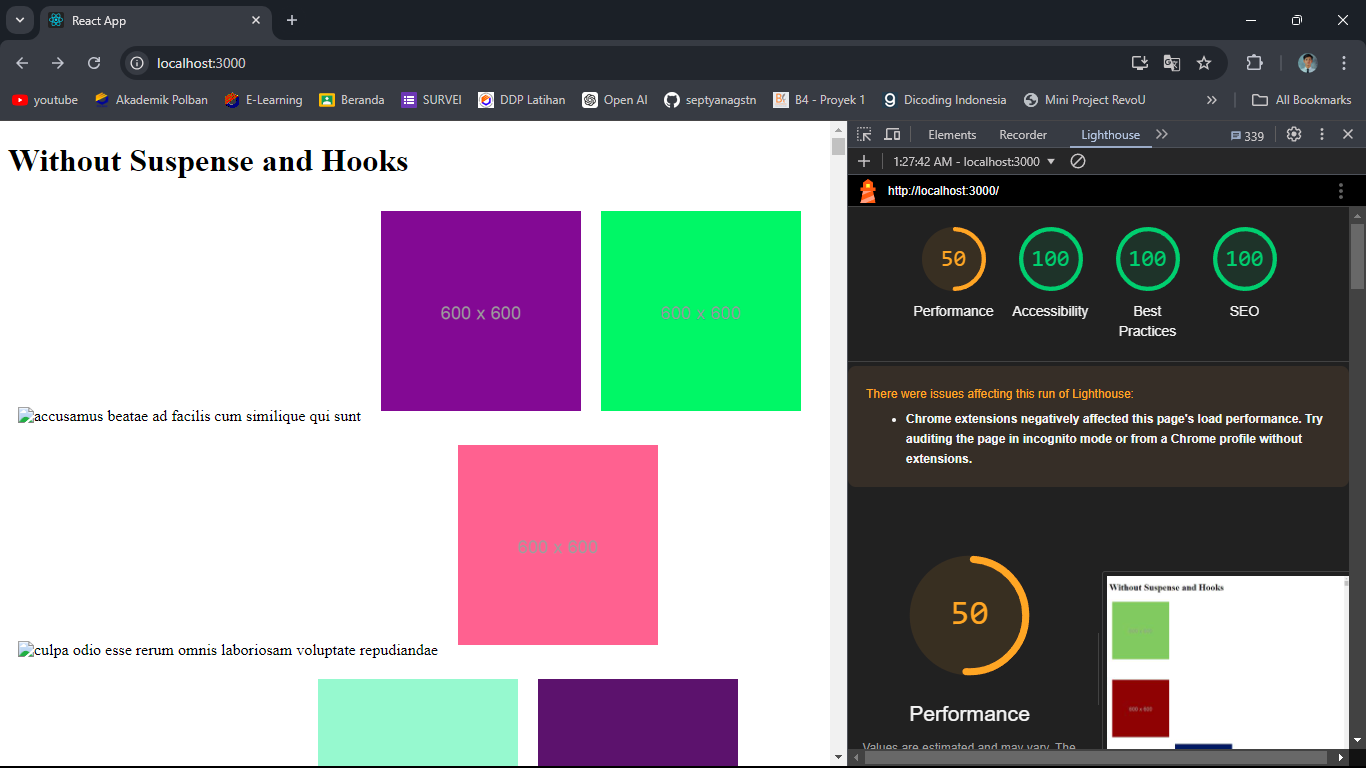
1. 100 Data



1. 1000 Data



1. 5000 Data



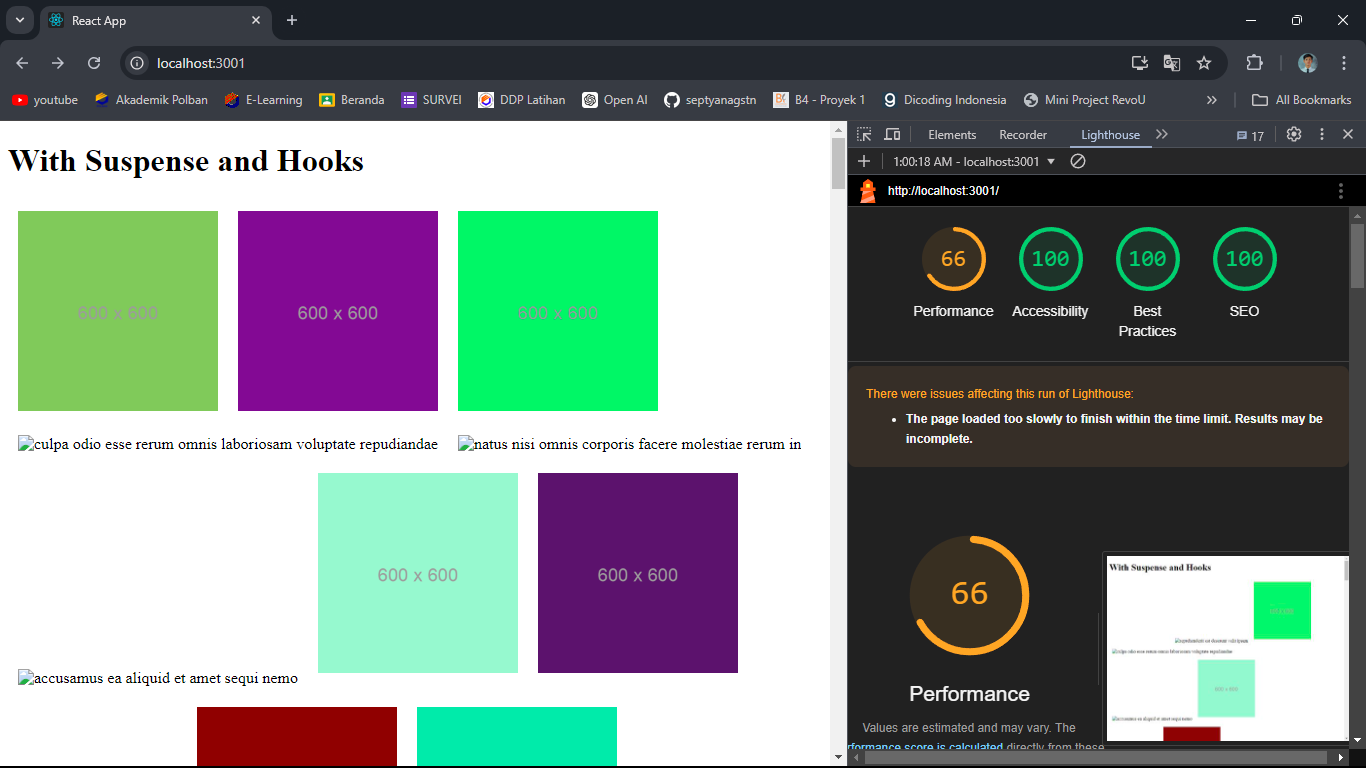
* **Dengan Suspense and Hooks**

1. 10 Data

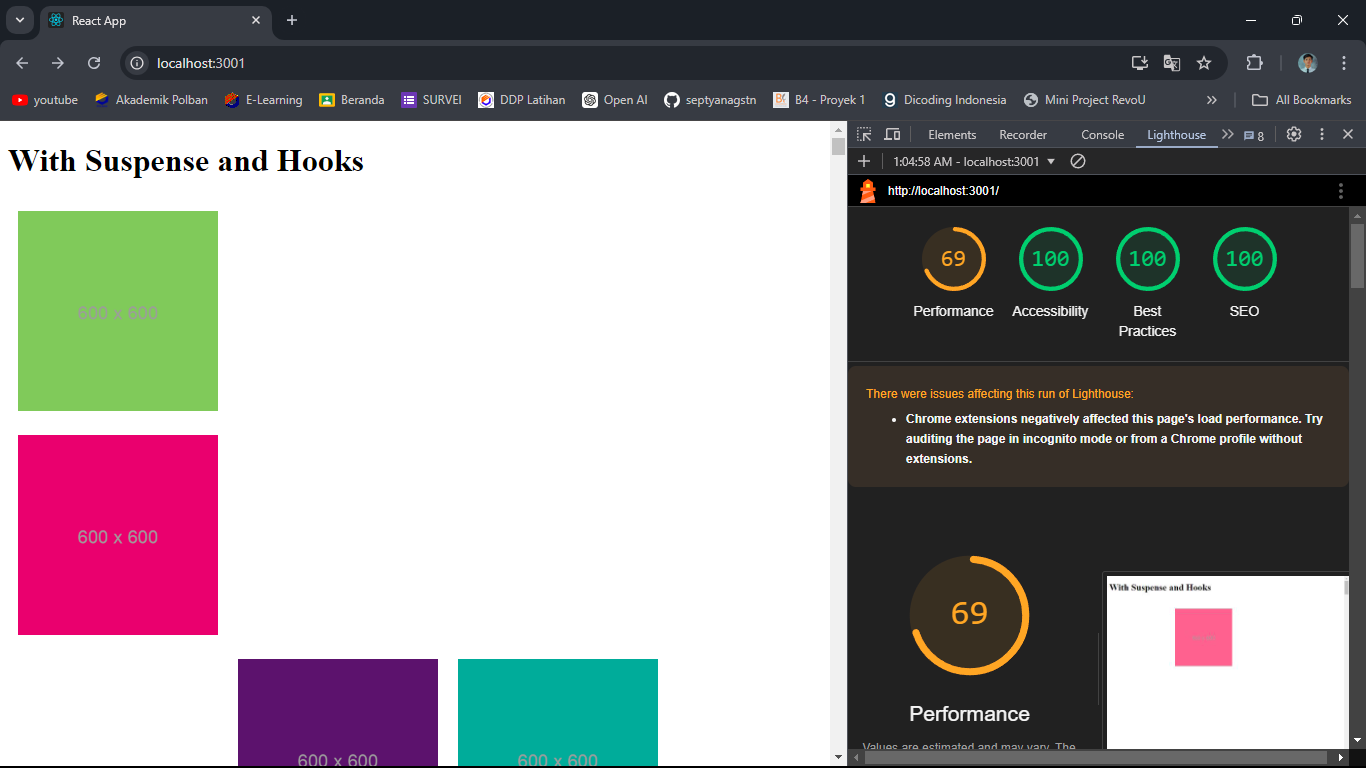
**A screenshot of a computer

Description automatically generated**

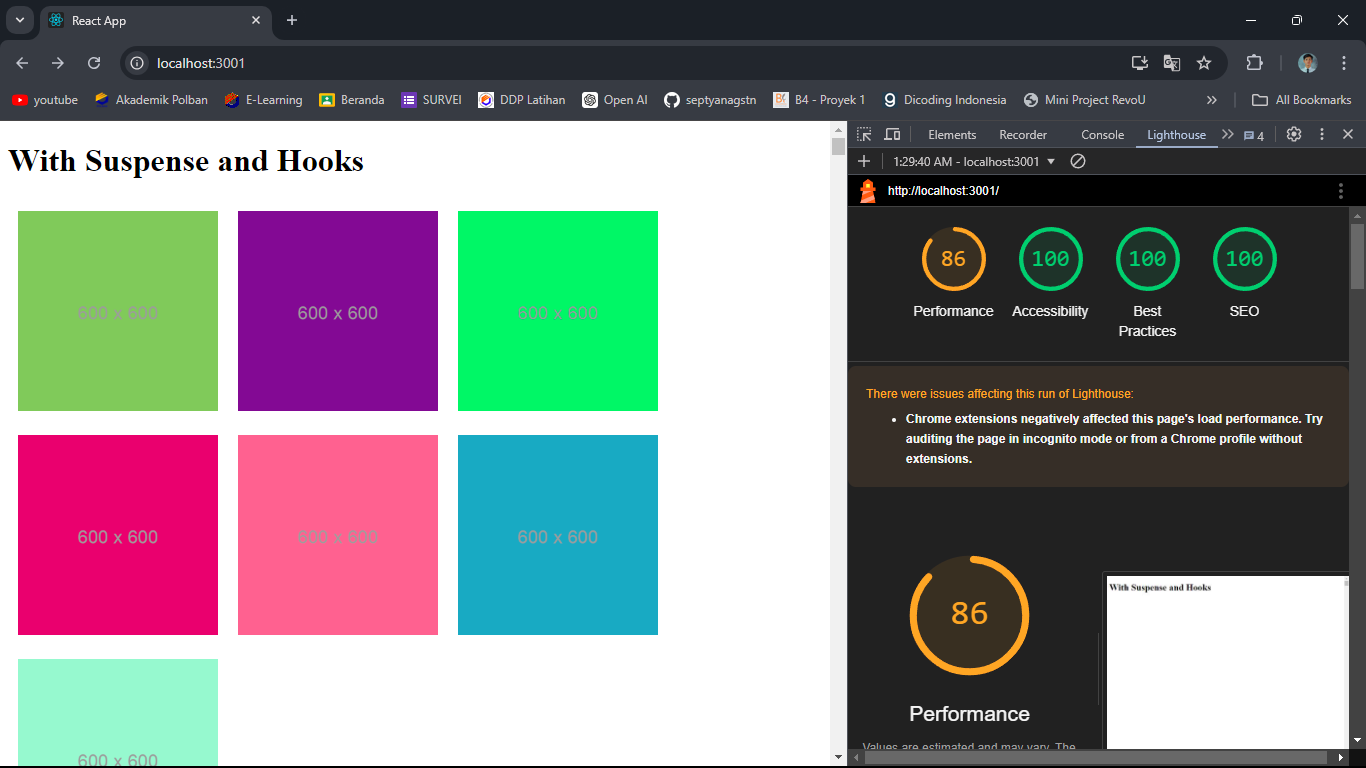
1. 100 Data

****

1. 1000 Data

****

1. 5000 Data

****

.

1. **Analisis Pros dan Cons**

Tanpa Suspense dan Hooks:

* Pros:
* Menggunakan metode tradisional yang lebih dikenal dan dapat diterapkan dalam berbagai versi React.
* Cons:
* Pengelolaan status loading dan error secara manual.
* Lebih banyak boilerplate dan potensi kode yang rumit.
* Waktu muat data mungkin lebih lama.

Dengan Suspense dan Hooks:

* Pros:
* Pengelolaan data fetching yang lebih otomatis dan efisien dengan React Query.
* Pengalaman pengguna yang lebih baik dengan Suspense dan pengelolaan caching otomatis.
* Kode lebih bersih dan ringkas.
* Cons:
* Memerlukan pemahaman tambahan tentang React Query dan Suspense.
* Menggunakan library eksternal yang mungkin memerlukan konfigurasi lebih lanjut.

1. **Kesimpulan**

Eksperimen menunjukkan bahwa menggunakan React Query dengan Suspense menawarkan keuntungan signifikan dalam hal performa dan pengalaman pengguna dibandingkan dengan metode tradisional yang menggunakan useEffect. Suspense dan React Query mengurangi waktu muat data, menyediakan pengelolaan status yang lebih baik, dan meningkatkan responsivitas aplikasi. Namun, pendekatan ini memerlukan pemahaman dan konfigurasi tambahan terkait dengan library eksternal.