README - Penjelasan Performa dan Penggunaan React dalam Aplikasi Kalkulator

Pengertian

Aplikasi ini adalah sebuah kalkulator untuk menghitung total durasi waktu yang dibutuhkan dalam sebuah latihan fisik, berdasarkan berbagai faktor seperti jumlah latihan, jumlah set, kecepatan latihan, dan durasi istirahat antar set. Fungsionalitas aplikasi ini mengandalkan React untuk mengelola state dan efek samping (side effects), serta menggunakan optimasi performa agar aplikasi dapat berjalan lebih efisien.

Dalam pembahasan ini, kita akan memfokuskan pada cara React mengelola **performa** dalam aplikasi, khususnya melalui penggunaan hooks seperti `useState`, `useEffect`, `useCallback`, dan `memo`. Ini adalah hal yang penting untuk dipahami agar aplikasi kita tetap cepat meskipun semakin kompleks.

Cara Berpikir React

React adalah pustaka JavaScript untuk membangun antarmuka pengguna (UI). Di dalam React, setiap perubahan state akan menyebabkan pembaruan ulang (re-render) dari komponen yang terpengaruh. Ini bisa menjadi masalah jika banyak komponen yang berubah setiap kali state diperbarui.

Untuk mengatasi masalah performa, React menyediakan beberapa mekanisme untuk **menghindari re-render yang tidak perlu**, seperti:

1. `memo` - Membantu React untuk menghindari re-render pada komponen jika props-nya tidak berubah.
2. `useCallback` - Menjaga agar fungsi tidak diciptakan ulang pada setiap render.
3. `useEffect` - Menjalankan efek samping berdasarkan perubahan state tertentu.

Analogi Sederhana

Bayangkan sebuah toko dengan beberapa karyawan yang menerima pelanggan. Ketika seorang pelanggan masuk, karyawan memeriksa apakah ada perubahan yang perlu dilakukan (misalnya, apakah ada informasi yang harus diperbarui). Namun, jika tidak ada perubahan yang relevan, karyawan tidak perlu mengulang pekerjaan mereka.

React bekerja dengan cara yang mirip. Ketika ada perubahan pada data, React akan “memeriksa” apakah perlu memperbarui UI. Namun, dengan optimisasi performa seperti `memo` dan `useCallback`, React hanya akan melakukan pembaruan jika benar-benar diperlukan, menghindari kerja yang berlebihan dan membuat aplikasi lebih cepat.

Penjelasan Code Tiap Baris

Berikut adalah penjelasan kode per komponen:

```
import { memo, useCallback, useEffect, useState } from "react";
import clickSound from "./ClickSound.m4a";
```

- `useState`: Hook untuk menyimpan dan mengubah state dalam komponen.
- `useEffect`: Hook untuk menangani efek samping, seperti pembaruan DOM atau pemanggilan API.
- `memo`: Fungsi HOC (Higher Order Component) untuk mencegah re-render jika props tidak berubah.
- `useCallback`: Hook untuk menjaga fungsi agar tidak diciptakan ulang pada setiap render.
- `clickSound`: Impor file suara untuk digunakan dalam aplikasi.

```
function Calculator({ workouts, allowSound }) {
```

- `Calculator`: Ini adalah komponen utama aplikasi yang menerima props `workouts` dan `allowSound`.

```
const [number, setNumber] = useState(workouts.at(0).numExercises);
const [sets, setSets] = useState(3);
const [speed, setSpeed] = useState(90);
const [durationBreak, setDurationBreak] = useState(5);
const [duration, setDuration] = useState(0);
```

- `useState` **hooks** digunakan untuk mendeklarasikan state di dalam komponen ini, seperti jumlah latihan, jumlah set, kecepatan latihan, durasi istirahat, dan total durasi.

```
useEffect(() => {
  setDuration((number * sets * speed) / 60 + (sets - 1) * durationBreak);
}, [number, sets, speed, durationBreak]);
```

- `useEffect` ini menjalankan pembaruan durasi (`setDuration`) setiap kali salah satu nilai yang terkait dengan durasi berubah, seperti `number`, `sets`, `speed`, dan `durationBreak`.

```
useEffect(
  function () {
    const playSound = function () {
      if (!allowSound) return;
      const sound = new Audio(clickSound);
      sound.play();
    };
    playSound();
  },
  [duration, allowSound]
);
```

- `useEffect` **kedua** bertugas memutar suara (jika `allowSound` diizinkan) setiap kali durasi berubah. Di sini, kita menggunakan efek samping untuk memutar audio. Efek ini hanya dijalankan ketika `duration` atau `allowSound` berubah.

```
useEffect(
  function () {
    document.title = `Your ${number}-exercise workout`;
  },
  [number]
);
```

- **`useEffect` ketiga**: Setiap kali `number` berubah, judul dokumen (tab browser) akan diperbarui, menunjukkan jumlah latihan yang sedang dipilih.

```
const mins = Math.floor(duration);
const seconds = (duration - mins) * 60;
```

- Menghitung menit dan detik dari `duration` untuk ditampilkan.

```
function handleInc() {
  setDuration((duration) => Math.floor(duration) + 1);
}
function handleDec() {
  setDuration((duration) => (duration > 1 ? Math.ceil(duration) - 1 : 0));
}
```

- Fungsi **`handleInc`** dan **`handleDec`** digunakan untuk menambah atau mengurangi durasi total dengan satu detik.

```
  return (
    <>
      <form>
        <div>
          <label>Type of workout</label>
          <select value={number} onChange={(e) =>
setNumber(+e.target.value)}>
            {workouts.map((workout) => (
              <option value={workout.numExercises} key={workout.name}>
                {workout.name} ({workout.numExercises} exercises)
              </option>
            ))}
          </select>
        </div>
        <div>
          <label>How many sets?</label>
          <input
            type="range"
            min="1"
            max="5"
            value={sets}
```

```
            onChange={(e) => setSets(e.target.value)}
          />
          <span>{sets}</span>
        </div>
        <div>
          <label>How fast are you?</label>
          <input
            type="range"
            min="30"
            max="180"
            step="30"
            value={speed}
            onChange={(e) => setSpeed(e.target.value)}
          />
          <span>{speed} sec/exercise</span>
        </div>
        <div>
          <label>Break length</label>
          <input
            type="range"
            min="1"
            max="10"
            value={durationBreak}
            onChange={(e) => setDurationBreak(e.target.value)}
          />
          <span>{durationBreak} minutes/break</span>
        </div>
      </form>
      <section>
        <button onClick={handleDec}>-</button>
        <p>
          {mins < 10 && "0"}
          {mins}:{seconds < 10 && "0"}
          {seconds}
        </p>
        <button onClick={handleInc}>+</button>
      </section>
    </>
  );
}
```

- **Formulir** untuk memilih jenis latihan, jumlah set, kecepatan, dan durasi istirahat.
- **Tombol** untuk meningkatkan atau mengurangi durasi total.
- `setNumber`, `setSets`, `setSpeed`, dan `setDurationBreak` digunakan untuk memperbarui state berdasarkan input pengguna.

```
export default memo(Calculator);
```

- **`memo`**: Membungkus komponen `Calculator` dengan `memo` untuk mencegah re-rendering jika props `workouts` dan `allowSound` tidak berubah. Ini adalah teknik optimisasi performa, menghindari

pembaruan ulang komponen yang tidak diperlukan.

Kesimpulan

- **Performa** adalah hal yang sangat penting dalam pengembangan aplikasi dengan React, terutama ketika aplikasi tumbuh lebih kompleks.
- Dengan menggunakan hooks seperti `useState`, `useEffect`, `memo`, dan `useCallback`, kita dapat mengoptimalkan pembaruan UI dan menghindari re-render yang tidak perlu.
- `memo` dan `useCallback` sangat membantu untuk menghindari pembaruan yang tidak perlu pada komponen dan fungsi, yang secara langsung meningkatkan performa aplikasi kita.