Penjelasan Kode React untuk Memoize dan Performa: `useReducer`, `useCallback`, dan `useContext`

Pengertian

Pada kode ini, kita membangun aplikasi React yang menggunakan berbagai hook untuk mengelola state dan memaksimalkan performa. Hook yang digunakan adalah:

1. `useReducer` untuk pengelolaan state yang lebih kompleks.
2. `useCallback` untuk memoization fungsi agar tidak dibuat ulang setiap kali render.
3. `useContext` untuk berbagi state antar komponen tanpa perlu prop drilling.

Pada aplikasi ini, kita berfokus pada pengelolaan data kota, seperti memuat data kota, membuat kota baru, dan menghapus kota, serta bagaimana kita memaksimalkan performa dengan memoization.

Cara Berpikir React

React bekerja dengan konsep **komponen yang terpisah** dan **state yang dikelola**. Komponen di React hanya akan merender ulang jika state atau props-nya berubah. Ini adalah konsep **reactivity** di React. React menggunakan Virtual DOM untuk melakukan pembandingan antara state lama dan baru, sehingga hanya elemen yang berubah yang akan diperbarui.

Untuk memaksimalkan performa, kita perlu menghindari perhitungan ulang yang tidak perlu, dan salah satu cara untuk melakukannya adalah dengan **memoization**. Memoization adalah teknik untuk menyimpan hasil dari fungsi agar jika fungsi tersebut dipanggil dengan parameter yang sama, kita bisa mengembalikan hasil yang sudah disimpan tanpa perlu menghitung ulang.

Analogi Sederhana

Bayangkan Anda memiliki sebuah mesin kalkulator yang memerlukan waktu lama untuk melakukan perhitungan besar. Jika Anda memanggil mesin kalkulator dengan masalah yang sama berulang kali, alih-alih membiarkan mesin kalkulator menghitungnya lagi, Anda memutuskan untuk **mengingat** hasil dari perhitungan tersebut dan langsung memberikan hasil yang sudah dihitung sebelumnya.

Begitu pula di React, kita "menghitung ulang" state atau fungsi hanya jika diperlukan, dan jika tidak, kita menggunakan hasil yang sudah ada untuk menghindari pekerjaan berlebihan.

Penjelasan Kode Tiap Baris

1. `useReducer` untuk Mengelola State

```
const [{ cities, isLoading, currentCity, error }, dispatch] = useReducer(
  reducer,
  initialState
);
```

- **`useReducer`** adalah hook untuk pengelolaan state yang lebih kompleks, dengan menggantikan `useState`. Ini berguna ketika state kita memiliki beberapa nilai atau aksi yang saling berhubungan.
- `reducer` adalah fungsi yang menangani perubahan state berdasarkan aksi yang dilakukan.
- `dispatch` digunakan untuk mengirimkan aksi yang akan mengubah state.

2. `useEffect` untuk Fetching Data

```
useEffect(function () {
  async function fetchCities() {
    dispatch({ type: "loading" });
    try {
      const res = await fetch(`${BASE_URL}/cities`);
      const data = await res.json();
      dispatch({ type: "cities/loaded", payload: data });
    } catch {
      dispatch({
        type: "rejected",
        payload: "There was an error loading cities.....",
      });
    }
  }
  fetchCities();
}, []);
```

- **`useEffect`** menjalankan kode untuk mengambil data kota dari API saat pertama kali komponen direnderer.
- `dispatch({ type: "loading" })` mengubah state untuk menandakan bahwa data sedang dimuat.
- Setelah data berhasil diambil, kita mengirimkan `dispatch({ type: "cities/loaded" })` untuk memperbarui state dengan data yang baru.

3. `useCallback` untuk Memoize Fungsi

```
const getCity = useCallback(
  async function getCity(id) {
    if (Number(id) === currentCity.id) return;

    dispatch({ type: "loading" });
    try {
      const res = await fetch(`${BASE_URL}/cities/${id}`);
      const data = await res.json();
      dispatch({ type: "city/loaded", payload: data });
    } catch {
      dispatch({
        type: "rejected",
        payload: "There was an error loading the city.....",
      });
    }
  },
```

```
    [currentCity.id]
  );
```

- **`useCallback`** digunakan untuk memastikan bahwa fungsi `getCity` tidak dibuat ulang setiap kali komponen dirender ulang, kecuali `currentCity.id` berubah.
- Tanpa `useCallback`, setiap kali komponen dirender ulang, fungsi `getCity` akan dibuat ulang, yang dapat memperburuk performa terutama jika fungsi tersebut dipassing ke komponen anak yang tidak memerlukannya untuk berubah.

4. `useContext` untuk Berbagi State

```
function useCities() {
  const context = useContext(CitiesContext);
  if (context === undefined)
    throw new Error("CitiesContext was used outside the CitiesProvider");
  return context;
}
```

- **`useContext`** digunakan untuk mengakses nilai dari `CitiesContext`. Ini memungkinkan kita untuk berbagi state antara komponen-komponen dalam aplikasi tanpa harus mengoper props secara langsung.
- Jika `useCities` dipanggil di luar `CitiesProvider`, maka akan menghasilkan error, karena konteks hanya bisa digunakan dalam komponen yang berada dalam `CitiesProvider`.

Kesimpulan

- **`useReducer`** sangat berguna untuk mengelola state yang lebih kompleks dengan banyak aksi.
- **`useEffect`** digunakan untuk menjalankan efek samping, seperti mengambil data dari API, setelah komponen dirender.
- **`useCallback`** adalah teknik memoization yang membantu kita untuk menghindari pembuatan fungsi baru yang tidak perlu pada setiap render, yang dapat meningkatkan performa aplikasi.
- **`useContext`** memungkinkan kita untuk berbagi state secara global antar komponen tanpa perlu mengoper props secara manual.

Dengan menggabungkan teknik-teknik ini, aplikasi React kita menjadi lebih efisien dan terstruktur dengan baik, sehingga memudahkan pengelolaan data dan meningkatkan performa aplikasi terutama pada aplikasi yang membutuhkan interaksi dengan data dari API.