Pemrograman Web Modern

Insan Taufik, S.Kom., M.Kom.

Life Cycle dan Side Effects (useEffect Hook)

Sub-Materi

Konsep Component Life Cycle (Mounting, Updating, Unmounting). Pengenalan hook useEffect untuk menangani side effects (misalnya fetching data, manipulasi DOM). Dependensi array pada useEffect.

Tujuan Pembelajaran (Capaian)

Memahami kapan dan bagaimana menjalankan kode side effect (di luar rendering) pada waktu yang tepat.

Apa itu Component Life Cycle?

Definisi:

Lifecycle adalah siklus hidup komponen React dari dibuat, diperbarui, hingga dihancurkan.

Tiga Fase Utama:

1. Mounting - Komponen dibuat dan dimasukkan ke DOM
2. Updating - Komponen diperbarui karena perubahan props atau state
3. Unmounting - Komponen dihapus dari DOM

Analoginya:

Seperti manusia: Lahir (Mount) → Tumbuh & Berubah (Update) → Meninggal (Unmount)

Life Cycle dalam Diagram

MOUNTING
↓
constructor()
↓
render()
↓
React memperbarui DOM
↓
componentDidMount()
↓
UPDATING (saat props/state berubah)
↓
render()
↓
React memperbarui DOM
↓
componentDidUpdate()
↓
UNMOUNTING
↓
componentWillUnmount()

Catatan: Diagram di samping untuk class components. Untuk functional components, kita menggunakan useEffect.

Sintaks Dasar useEffect

```
import { useEffect } from 'react';

const MyComponent = () => {
  useEffect(() => {
    // Kode side effect di sini

    return () => {
      // Cleanup function (opsional)
    };
  }, [dependencies]); // Dependencies array
};
```

Penjelasan Struktur:

- Callback function: Berisi kode side effect yang akan dijalankan
- Dependencies array: Menentukan kapan effect dijalankan
- Cleanup function: Untuk membersihkan effect sebelum komponen unmount atau re-run effect

Jenis-jenis useEffect Berdasarkan Dependencies

```
// 1. Tanpa dependencies - run setelah setiap render
useEffect(() => {
  console.log('Ini dijalankan setelah setiap render');
});

// 2. Empty dependencies - run sekali setelah mount
useEffect(() => {
  console.log('Ini dijalankan sekali setelah mount');
}, []);

// 3. Dengan dependencies - run ketika dependency berubah
useEffect(() => {
  console.log('Count berubah:', count);
}, [count]);
```

Best Practice:
Selalu sertakan dependencies yang digunakan di dalam effect ke dalam dependencies array.

useEffect untuk Fetching Data

```jsx
const UserProfile = ({ userId }) => {
  const [user, setUser] = useState(null);

  useEffect(() => {
    const fetchUser = async () => {
      const response = await fetch(`/api/users/${userId}`);
      const userData = await response.json();
      setUser(userData);
    };

    fetchUser();
  }, [userId]); // Re-fetch ketika userId berubah

  return <div>{user?.name}</div>;
};
```

Penting:

- Gunakan async/function di dalam effect, bukan effect function itu sendiri
- Sertakan dependencies yang diperlukan

Cleanup dalam useEffect

```jsx
const Timer = () => {
  const [time, setTime] = useState(0);

  useEffect(() => {
    const interval = setInterval(() => {
      setTime(prev => prev + 1);
    }, 1000);

    // Cleanup function
    return () => {
      clearInterval(interval);
    };
  }, []); // Empty dependencies - run sekali

  return <div>Time: {time}s</div>;
};
```

Kapan cleanup dijalankan?

- Sebelum komponen unmount
- Sebelum effect dijalankan lagi (jika dependencies berubah)

Common Use Cases useEffect

```
// 1. Event listeners
useEffect(() => {
  const handleResize = () => console.log(window.innerWidth);
  window.addEventListener('resize', handleResize);

  return () => window.removeEventListener('resize', handleResize);
}, []);

// 2. Document title update
useEffect(() => {
  document.title = `You have ${count} messages`;
}, [count]);

// 3. LocalStorage synchronization
useEffect(() => {
  localStorage.setItem('userData', JSON.stringify(userData));
}, [userData]);
```

Ringkasan dan Best Practices

Yang telah dipelajari:

- ✓ Konsep component life cycle
- ✓ Penggunaan useEffect untuk side effects
- ✓ Dependencies array dan pengaruhnya
- ✓ Cleanup function

Best Practices:

- Jangan lupa dependencies array
- Gunakan cleanup untuk mencegah memory leaks
- Pisahkan concerns dengan multiple useEffect
- Hindari infinite loops dengan dependencies yang tepat