Pemrograman Web Modern

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

Manajemen State Lanjutan (Context API)

Sub-Materi

Batasan Prop Drilling.
Pengenalan Context API sebagai alternatif untuk state management sederhana.
Membuat Provider dan Consumer/useContext.

Tujuan Pembelajaran (Capaian)

Mampu mengelola state global pada skala aplikasi kecil hingga menengah tanpa Prop Drilling.

Masalah Prop Drilling

Apa itu Prop Drilling?

- Proses mengoper props melalui banyak level komponen
- Props dilewatkan ke komponen yang tidak membutuhkannya
- Hanya untuk sampai ke komponen anak yang dalam

Contoh Prop Drilling:

```
// Komponen A (punya data)
<KomponenB data={data} />

// Komponen B (tidak butuh data, hanya oper)
<KomponenC data={data} />

// Komponen C (butuh data)
<div>{data}</div>
```

Masalah Prop Drilling:

- Kode sulit dibaca dan dipelihara
- Perubahan props mempengaruhi banyak komponen
- Komponen menjadi tightly coupled

Pengenalan Context API

Apa itu Context API?

- Fitur built-in React untuk state management
- Alternatif untuk Prop Drilling
- Membuat state global yang bisa diakses oleh semua komponen

Kapan menggunakan Context API?

- State yang digunakan oleh banyak komponen
- Aplikasi kecil hingga menengah
- Theme, user authentication, bahasa, dll

Kapan TIDAK menggunakan Context API?

- Data yang sering berubah (performance issue)
- Aplikasi sangat besar (gunakan Redux/Zustand)

Konsep Context API

Three Main Parts:

1. createContext() - Membuat context object
2. Provider - Komponen yang menyediakan data
3. Consumer / useContext - Komponen yang mengonsumsi data

Data Flow:

createContext() → Provider → Consumer/useContext

Membuat Context

Syntax createContext:

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

const MyContext = createContext(defaultValue);
```

Penjelasan:

- createContext membuat context object baru
- defaultValue digunakan ketika tidak ada Provider
- Context object memiliki Provider dan Consumer

Contoh:

```
// Membuat Theme Context
const ThemeContext = createContext('light');

// Membuat User Context
const UserContext = createContext(null);
```

Provider Component

Fungsi Provider:

- Membungkus komponen yang butuh akses data
- Menyediakan value yang bisa diakses children

Syntax Provider:

```
<MyContext.Provider value={/* some value */}>
  <ChildComponents />
</MyContext.Provider>
```

Contoh Implementasi:

```
function App() {
  const [theme, setTheme] = useState('light');

  return (
    <ThemeContext.Provider value={{ theme, setTheme }}>
      <Header />
      <Main />
      <Footer />
    </ThemeContext.Provider>
  );
}
```

Mengonsumsi Data dengan useContext

useContext Hook:

- Cara modern untuk mengakses context
- Lebih sederhana daripada Consumer

Syntax useContext:

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

function MyComponent() {
  const contextValue = useContext(MyContext);
  return <div>{contextValue}</div>;
}
```

Contoh Penggunaan:

```
function Header() {
  const { theme, setTheme } = useContext(ThemeContext);

  return (
    <header className={theme}>
      <button onClick={() => setTheme('dark')}>
        Toggle Theme
      </button>
    </header>
  );
}
```

Consumer Component (Class-style)

Alternative to useContext:

- Cara tradisional menggunakan Consumer
- Berguna untuk class components

Syntax Consumer:

```
<MyContext.Consumer>
  {value => /* render sesuatu berdasarkan value */}
</MyContext.Consumer>
```

Contoh:

```
function ThemeDisplay() {
  return (
    <ThemeContext.Consumer>
      {({ theme }) => (
        <div>Current theme: {theme}</div>
      )}
    </ThemeContext.Consumer>
  );
}
```

Best Practices Context API

1. Pisahkan Context Berdasarkan Feature

```
// Jangan: satu context untuk semua
// Baik: multiple context specific
const UserContext = createContext();
const ThemeContext = createContext();
const CartContext = createContext();
```

2. Gunakan Custom Provider

```
function UserProvider({ children }) {
  const [user, setUser] = useState(null);

  return (
    <UserContext.Provider value={{ user, setUser }}>
      {children}
    </UserContext.Provider>
  );
}
```

3. Optimasi Performance

- Hindari frequent updates di context value
- Pisahkan state dan setter di context berbeda

Ringkasan

Yang telah dipelajari:

- ✓ Masalah Prop Drilling dan solusinya
- ✓ Konsep Context API
- ✓ Membuat Context dan Provider
- ✓ Mengonsumsi data dengan useContext
- ✓ Best practices Context API

Keuntungan Context API:

- Tidak perlu Prop Drilling
- Code lebih clean dan maintainable
- State management yang sederhana