📆 Hari 1 (Senin) - Mengelola State Keranjang Belanja (Cart)

🎯 Tujuan Pembelajaran Hari Ini

- Memahami konsep dasar dan pentingnya state keranjang belanja (cart) di aplikasi frontend e-commerce.
- Mengeksplorasi pilihan pendekatan state management untuk mengelola data keranjang.
- Menentukan struktur data yang efektif untuk menyimpan item di keranjang.
- Mampu mengimplementasikan logika untuk menambahkan produk ke keranjang, termasuk menangani produk duplikat.

📚 Materi Inti (2 Jam)

1. Konsep State Keranjang Belanja di Frontend

Dalam aplikasi e-commerce, keranjang belanja adalah fitur krusial yang memungkinkan user mengumpulkan produk yang ingin mereka beli sebelum melanjutkan ke proses checkout. Di sisi frontend, kita perlu menyimpan dan mengelola daftar produk yang ada di keranjang user. Data ini bersifat *stateful*, artinya bisa berubah seiring interaksi user (menambah, menghapus, mengubah kuantitas).

State keranjang belanja ini harus bisa diakses oleh berbagai komponen di aplikasi, misalnya:

- Komponen daftar produk atau detail produk (untuk tombol "Tambah ke Keranjang").
- Komponen header/navbar (untuk menampilkan jumlah item di keranjang).
- Halaman keranjang itu sendiri (untuk menampilkan detail item).
- Halaman checkout (untuk menampilkan ringkasan pesanan).

Oleh karena itu, manajemen state keranjang ini memerlukan pendekatan yang tepat agar data konsisten dan mudah diakses di seluruh aplikasi.

2. Memilih Pendekatan State Management untuk Cart

Ada beberapa cara untuk mengelola state keranjang di aplikasi React:

- **State Lokal (`useState`):** State keranjang disimpan di komponen parent terdekat yang membutuhkan data tersebut, lalu diteruskan ke komponen child melalui props. Ini cocok untuk aplikasi sangat sederhana, tetapi bisa menyebabkan *prop drilling* (meneruskan prop melalui banyak level komponen) jika state dibutuhkan di komponen yang jauh.
- **Context API:** Menggunakan React Context untuk membuat state global yang bisa diakses oleh komponen mana pun di bawah `Provider`-nya. Ini adalah solusi bawaan React yang baik untuk state global yang tidak terlalu kompleks dan sering berubah. Kita sudah menggunakan ini untuk state autentikasi di Minggu 8.
- **Library State Management Eksternal (Redux Toolkit, Zustand, dll.):** Library pihak ketiga yang menyediakan struktur dan pola yang lebih canggih untuk mengelola state global, terutama untuk

aplikasi besar dengan state yang kompleks dan banyak interaksi. Mereka seringkali menawarkan fitur tambahan seperti devtools, middleware, dan optimasi performa.

Untuk minggu ini, kita akan fokus menggunakan **Context API** karena sudah cukup memadai untuk kebutuhan keranjang belanja di proyek e-commerce sederhana kita dan agar Anda semakin familiar dengan pola ini.

3. Struktur Data untuk Menyimpan Item di Keranjang

Bagaimana kita merepresentasikan item-item di keranjang dalam state kita? Struktur data yang umum digunakan adalah sebuah **Array of Objects**. Setiap objek dalam array merepresentasikan satu item unik di keranjang dan berisi informasi yang relevan.

Informasi minimum yang biasanya disimpan untuk setiap item di keranjang adalah:

- **ID Produk:** Identifier unik untuk produk tersebut.
- **Detail Produk:** Informasi seperti nama, harga, gambar, dll. (Bisa disimpan langsung atau hanya ID-nya, lalu detail lengkap diambil saat menampilkan keranjang).
- **Kuantitas:** Jumlah produk tersebut yang ada di keranjang.

Contoh struktur state keranjang:

```
[
  {
    product: { // Objek produk lengkap atau sebagian info penting
      id: 'prod-123',
      name: 'Laptop Gaming',
      price: 15000000,
      imageUrl: '/images/laptop.jpg'
    },
    quantity: 1
  },
  {
    product: {
      id: 'prod-456',
      name: 'Mouse Wireless',
      price: 300000,
      imageUrl: '/images/mouse.jpg'
    },
    quantity: 2
  }
]
```

Memilih menyimpan objek produk lengkap di state keranjang memudahkan saat menampilkan detail di halaman keranjang tanpa perlu mencari data produk lagi. Namun, jika objek produk sangat besar, menyimpan hanya ID dan kuantitas, lalu mencari detail produk saat dibutuhkan (misalnya dari daftar produk yang sudah di-fetch atau endpoint API terpisah), bisa lebih efisien.

Untuk latihan, kita akan menyimpan objek produk lengkap beserta kuantitasnya.

4. Menambahkan Item ke Keranjang (`addToCart`)

Fungsi `addToCart` adalah logika inti saat user mengklik tombol "Tambah ke Keranjang". Fungsi ini akan menerima objek produk yang ingin ditambahkan dan memperbarui state keranjang.

Ada dua skenario utama saat menambahkan item:

1. **Produk Belum Ada di Keranjang:** Produk baru ditambahkan ke array state keranjang dengan kuantitas awal 1.
2. **Produk Sudah Ada di Keranjang:** Kuantitas produk yang sudah ada di keranjang ditingkatkan (misalnya, ditambah 1).

Kita perlu mengimplementasikan logika untuk mengecek apakah produk sudah ada di keranjang berdasarkan ID produknya.

Contoh implementasi fungsi `addToCart` (menggunakan Context API):-

```
// src/contexts/CartContext.js (Contoh di dalam CartProvider)

// ... import createContext, useContext, useState ...

const CartContext = createContext(null);

export const CartProvider = ({ children }) => {
  const [cartItems, setCartItems] = useState([]);

  const addToCart = (product) => {
    // Cek apakah produk sudah ada di keranjang
    const existingItemIndex = cartItems.findIndex(
      (item) => item.product.id === product.id
    );

    if (existingItemIndex > -1) {
      // Jika produk sudah ada, update kuantitasnya
      const updatedCartItems = [...cartItems]; // Buat salinan array
(immutability!)
      updatedCartItems[existingItemIndex].quantity += 1;
      setCartItems(updatedCartItems);
    } else {
      // Jika produk belum ada, tambahkan sebagai item baru dengan
kuantitas 1
      setCartItems([...cartItems, { product, quantity: 1 }]); // Tambahkan
item baru (immutability!)
    }
  };

  // ... fungsi lain seperti updateQuantity, removeFromCart, dll. (akan
dibuat besok)

  const value = {
    cartItems,
    addToCart,
```

```
    // ... fungsi lain
  };

  return <CartContext.Provider value={value}>{children}
</CartContext.Provider>;
};

export const useCart = () => {
  const context = useContext(CartContext);
  if (context === undefined) {
    throw new Error('useCart must be used within a CartProvider');
  }
  return context;
};
```

Penting: Saat mengupdate state array atau objek di React, kita harus selalu mengikuti prinsip **Immutability**. Artinya, jangan memodifikasi state yang sudah ada secara langsung. Selalu buat salinan baru dari state yang ingin diubah, lalu terapkan perubahan pada salinan tersebut, dan gunakan fungsi `set...` (misal `setCartItems`) dengan salinan baru tersebut. Menggunakan spread operator (`...`) adalah cara umum untuk membuat salinan array atau objek.

🏋️ Praktik Mandiri (8 Jam)

1. **Lanjutkan proyek e-commerce Anda.** Pastikan Anda sudah memiliki halaman daftar produk atau detail produk yang menampilkan informasi produk.
2. **Buat Context API untuk Cart:** Buat file baru `src/contexts/CartContext.js`. Salin kode dasar untuk `CartContext`, `CartProvider`, dan hook `useCart` seperti contoh di atas.
3. **Integrasikan `CartProvider`:** Di file entry point aplikasi Anda (misalnya `src/index.js` atau `src/App.js`), impor `CartProvider` dan bungkus komponen root (`<App />`) dengannya. Pastikan `CartProvider` berada di dalam `AuthProvider` jika Anda sudah mengimplementasikannya di Minggu 8.

   ```
   // src/index.js
   import React from 'react';
   import ReactDOM from 'react-dom/client';
   import App from './App';
   import { AuthProvider } from './contexts/AuthContext'; // Jika ada
   import { CartProvider } from './contexts/CartContext'; // Import
   CartProvider

   const root = ReactDOM.createRoot(document.getElementById('root'));
   root.render(
     <React.StrictMode>
       <AuthProvider> {/* AuthProvider di luar CartProvider jika perlu
   */}
         <CartProvider> {/* Bungkus App dengan CartProvider */}
           <App />
         </CartProvider>
       </AuthProvider>
   ```

```
    </React.StrictMode>
);
```

4. **Implementasikan fungsi `addToCart`:** Lengkapi fungsi `addToCart` di `CartContext.js` sesuai logika yang dijelaskan (cek produk duplikat, update kuantitas atau tambah item baru).
5. **Integrasikan `addToCart` di UI:** Di komponen daftar produk atau detail produk, impor hook `useCart()`. Dapatkan fungsi `addToCart` dari hook tersebut.

```
// Contoh di ProductDetailPage.jsx
import React, { useEffect, useState } from 'react';
import { useParams } from 'react-router-dom';
import { useCart } from '../contexts/CartContext'; // Import useCart
// ... import axios atau fetch ...

const ProductDetailPage = () => {
  const { id } = useParams();
  const [product, setProduct] = useState(null);
  const { addToCart } = useCart(); // Dapatkan addToCart dari Context
  // ... state loading, error ...

  useEffect(() => {
    // ... fetch detail produk berdasarkan id ...
    // setProduct(fetchedProduct);
  }, [id]);

  const handleAddToCart = () => {
    if (product) {
      addToCart(product); // Panggil fungsi addToCart dengan objek
produk
      alert(`${product.name} ditambahkan ke keranjang!`); //
Notifikasi sederhana
    }
  };

  if (!product) return <div>Loading...</div>; // Atau tampilkan error

  return (
    <div>
      <h1>{product.name}</h1>
      <p>Harga: Rp {product.price.toLocaleString()}</p>
      {/* ... detail produk lainnya ... */}
      <button onClick={handleAddToCart}>Tambah ke Keranjang</button>
    </div>
  );
};

export default ProductDetailPage;
```

6. **Uji coba penambahan item:** Buka halaman daftar produk atau detail produk. Klik tombol "Tambah ke Keranjang" beberapa kali untuk produk yang sama dan produk yang berbeda. Meskipun belum ada

tampilan keranjang, Anda bisa menggunakan React DevTools untuk memeriksa state `cartItems` di `CartContext` dan memverifikasi bahwa item ditambahkan dengan benar dan kuantitas diperbarui untuk item duplikat.

💡 Tips Belajar Tambahan

- **React DevTools:** Gunakan ekstensi React Developer Tools di browser Anda untuk memeriksa state `cartItems` di `CartContext`. Ini sangat membantu untuk debugging.
- **Immutability:** Pastikan Anda selalu membuat salinan baru dari array `cartItems` saat menambah atau mengubah item. Jangan pernah memodifikasi array state secara langsung.
- **ID Unik:** Pastikan setiap produk memiliki ID unik yang digunakan untuk identifikasi di keranjang.
- **Notifikasi:** Untuk pengalaman user yang lebih baik, tambahkan notifikasi (misalnya menggunakan library seperti `react-toastify`) saat item berhasil ditambahkan ke keranjang.

🔗 Referensi Tambahan

- React Documentation - Context
- React Documentation - useContext Hook
- React Documentation - useState Hook
- Working with Arrays in JavaScript (map, filter, findIndex, spread operator)

Hari ini kita telah meletakkan dasar untuk fitur keranjang belanja dengan memahami konsep state-nya dan mengimplementasikan logika penambahan item menggunakan Context API. Besok, kita akan fokus pada bagaimana menampilkan item-item yang sudah ada di keranjang dalam sebuah halaman khusus.