📆 Hari 4 (Kamis) - Halaman Detail Produk dan Pengiriman Header Autentikasi

🎯 Tujuan Pembelajaran Hari Ini

- Mampu membuat halaman detail produk yang menampilkan informasi lengkap satu produk.
- Mengambil ID produk dari URL menggunakan `react-router-dom`.
- Menggunakan `useEffect` dan `axios` untuk mengambil data detail produk dari API backend berdasarkan ID.
- Memahami konsep endpoint yang dilindungi (protected endpoints) di backend.
- Mampu mengirim token autentikasi (dari `localStorage` atau state) dalam header permintaan HTTP menggunakan `axios`.
- Mengimplementasikan penanganan loading dan error state untuk halaman detail produk.

📚 Materi Inti (2 Jam)

1. Membuat Halaman Detail Produk

- **Komponen `ProductDetailPage.jsx`:**
  - Komponen ini akan bertanggung jawab menampilkan detail lengkap dari satu produk.
  - Berbeda dengan `ProductItem` yang hanya menampilkan ringkasan dalam daftar, halaman detail akan menampilkan semua informasi yang tersedia untuk produk tersebut.
- **Mengambil ID Produk dari URL:**
  - Saat user mengklik produk dari daftar, kita akan menavigasikan mereka ke URL seperti `/products/:productId`.
  - Di komponen `ProductDetailPage`, kita perlu membaca nilai `:productId` dari URL untuk mengetahui produk mana yang harus diambil datanya.
  - Hook `useParams` dari `react-router-dom` digunakan untuk tujuan ini.

```
import React from 'react';
import { useParams } from 'react-router-dom';

function ProductDetailPage() {
  const { productId } = useParams(); // Mengambil nilai parameter
'productId' dari URL

  // ... logika fetching data menggunakan productId ...

  return (
    <div>
      <h1>Product Detail</h1>
      <p>Product ID from URL: {productId}</p>
      {/* ... tampilkan detail produk ... */}
    </div>
  );
}
```

```
export default ProductDetailPage;
```

- **Mengambil Data Detail Produk dengan `useEffect` dan `axios`:**
  - Sama seperti mengambil daftar produk, kita akan menggunakan `useEffect` untuk memanggil fungsi fetching data saat komponen mount.
  - Endpoint backend untuk detail produk biasanya `/api/products/:productId`.
  - Kita akan menggunakan `axios.get` ke endpoint ini, menyertakan `productId` yang diambil dari `useParams`.
  - **Dependency Array:** `useEffect` untuk fetching detail produk harus memiliki `productId` di dependency array-nya (`[productId]`). Ini memastikan bahwa jika ID produk di URL berubah (misalnya, user navigasi dari `/products/1` ke `/products/2`), effect akan dijalankan kembali untuk mengambil data produk yang baru.

```
// Di dalam ProductDetailPage.jsx
import React, { useEffect, useState } from 'react';
import { useParams } from 'react-router-dom';
import axios from 'axios';

function ProductDetailPage() {
  const { productId } = useParams();
  const [product, setProduct] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    async function fetchProductDetail() {
      try {
        setLoading(true); // Mulai loading
        setError(null); // Reset error
        // Mengambil data detail produk berdasarkan productId
        const response = await
axios.get(`/api/products/${productId}`);
        setProduct(response.data); // Update state dengan data produk
      } catch (err) {
        console.error('Error fetching product detail:', err);
        setError(err); // Update state jika ada error
      } finally {
        setLoading(false); // Selesai loading
      }
    }

    fetchProductDetail();

  }, [productId]); // Effect dijalankan ulang jika productId berubah

  if (loading) {
    return <div>Loading product details...</div>;
  }
```

```
  if (error) {
    return <div>Error loading product: {error.message}</div>;
  }

  if (!product) {
    return <div>Product not found.</div>; // Kasus jika data tidak
ditemukan setelah loading selesai
  }

  return (
    <div>
      <h1>{product.name}</h1>
      <img src={product.imageUrl} alt={product.name} style={{ width:
'200px' }} />
      <p>Price: ${product.price}</p>
      <p>{product.description}</p>
      {/* ... detail lainnya ... */}
    </div>
  );
}

export default ProductDetailPage;
```

2. Pengiriman Header Autentikasi

- **Konsep Endpoint yang Dilindungi (Protected Endpoints):**
  - Tidak semua endpoint API bisa diakses oleh siapa saja. Endpoint seperti "Get User Profile", "Create Order", "Update Product" (untuk admin) memerlukan user untuk login terlebih dahulu.
  - Backend akan memeriksa apakah permintaan datang dari user yang terautentikasi dan memiliki izin yang cukup.
  - Mekanisme paling umum adalah dengan memeriksa token autentikasi yang dikirim oleh frontend di setiap permintaan ke endpoint yang dilindungi.
- **Mengirim Token dalam Header Permintaan:**
  - Token autentikasi biasanya dikirim dalam header HTTP `Authorization` dengan skema `Bearer`.
  - Formatnya adalah `Authorization: Bearer <token_anda>`.
  - `axios` memudahkan penambahan header ini pada permintaan.

```
// Contoh mengirim token saat mengambil profil user
async function fetchUserProfile() {
  const token = localStorage.getItem('token'); // Ambil token dari
penyimpanan

  if (!token) {
    // User belum login, arahkan ke halaman login atau tampilkan pesan
    console.log('User not authenticated.');
    return;
  }

  try {
```

```
    const response = await axios.get('/api/users/profile', {
      headers: {
        Authorization: `Bearer ${token}` // Menambahkan header
Authorization
      }
    });
    console.log('User Profile:', response.data);
  } catch (err) {
    console.error('Error fetching profile:', err);
    // Tangani error, misalnya token expired atau tidak valid (status
401/403)
    if (err.response && (err.response.status === 401 ||
err.response.status === 403)) {
      console.log('Token invalid or expired. Please login again.');
      // Hapus token dan arahkan user ke halaman login
      localStorage.removeItem('token');
      // navigate('/login'); // Gunakan useNavigate jika di dalam
komponen React
    }
  }
}

// Panggil fungsi ini di useEffect pada komponen ProfilePage, misalnya
```

- **Menggunakan Interceptors `axios` untuk Otomatisasi:**
  - Menambahkan header `Authorization` secara manual di setiap permintaan bisa merepotkan.
  - `axios` interceptors memungkinkan kita menambahkan header ini secara otomatis ke setiap permintaan yang keluar.
  - Ini adalah pola yang lebih bersih dan DRY (Don't Repeat Yourself).

```
// Contoh setup interceptor (biasanya di file terpisah, misalnya
api.js)
import axios from 'axios';

const api = axios.create({
  baseURL: '/api', // Set base URL untuk semua request
  // timeout: 1000,
});

// Tambahkan request interceptor
api.interceptors.request.use(
  config => {
    const token = localStorage.getItem('token');
    if (token) {
      // Jika token ada, tambahkan header Authorization
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  },
  error => {
    // Lakukan sesuatu dengan request error
```

```
    return Promise.reject(error);
  }
);

// Tambahkan response interceptor (opsional, untuk menangani error
401/403 secara global)
api.interceptors.response.use(
  response => response,
  error => {
    // Lakukan sesuatu dengan response error
    if (error.response && (error.response.status === 401 ||
error.response.status === 403)) {
      console.log('Authentication error. Redirecting to login.');
      // Redirect user ke halaman login (perlu cara untuk mengakses
history/navigate di luar komponen)
      // Contoh sederhana: window.location.href = '/login';
      // Atau gunakan custom history object jika memakai react-router
v5 atau context/state management untuk v6
    }
    return Promise.reject(error);
  }
);

export default api; // Export instance axios yang sudah dikonfigurasi
```

- Setelah setup interceptor, Anda cukup menggunakan instance `api` ini untuk semua permintaan yang memerlukan autentikasi.

```
// Di komponen ProductDetailPage atau ProfilePage
import api from './api'; // Import instance axios yang sudah
dikonfigurasi

// ... di dalam useEffect ...
const response = await api.get(`/products/${productId}`); //
Menggunakan instance 'api'
// ... header Authorization otomatis ditambahkan jika token ada di
localStorage
```

3. Menangani Loading dan Error State

- Sama seperti fetching daftar produk, penting untuk mengelola state `loading` dan `error` saat mengambil data detail produk.
- Tampilkan indikator loading saat permintaan sedang berjalan.
- Tampilkan pesan error jika permintaan gagal.
- Tampilkan data produk setelah berhasil diambil.
- Pertimbangkan juga state jika produk dengan ID tersebut tidak ditemukan (backend merespons 404 Not Found).

🏋️ Praktik Mandiri (8 Jam)

1. **Konfigurasi Routing:** Pastikan Anda memiliki Route di `react-router-dom` untuk halaman detail produk, misalnya `/products/:productId`.
2. **Buat Komponen `ProductDetailPage.jsx`:**
   - Gunakan `useParams` untuk mendapatkan `productId` dari URL.
   - Gunakan `useState` untuk state `product`, `loading`, dan `error`.
   - Gunakan `useEffect` dengan dependency array `[productId]`.
   - Di dalam `useEffect`, buat fungsi `async` yang menggunakan `axios.get` untuk mengambil data dari dummy API detail produk (Anda mungkin perlu mensimulasikan endpoint ini atau menggunakan API publik yang mendukung detail item berdasarkan ID, contoh: `https://fakestoreapi.com/products/:id`).
   - Tangani respons sukses, error, dan state loading/error.
   - Tampilkan detail produk (nama, harga, deskripsi, gambar) jika data berhasil diambil.
3. **Tambahkan Link dari Daftar Produk:** Di komponen `ProductList` (dari Hari 2), ubah setiap item produk menjadi link (`<Link to={\`/products/${product.id}\`}>`) yang mengarah ke halaman detail produk.
4. **Implementasikan Pengiriman Header Autentikasi (Simulasi):**
   - Buat file konfigurasi `axios` terpisah (misalnya, `api.js`).
   - Buat instance `axios` menggunakan `axios.create()`.
   - Tambahkan request interceptor yang membaca token dari `localStorage` dan menambahkannya ke header `Authorization` jika token ada.
   - Gunakan instance `api` yang sudah dikonfigurasi ini untuk fetching data detail produk.
   - (Opsional) Simulasikan endpoint backend yang memerlukan autentikasi dan lihat apakah header `Authorization` terkirim.
5. **Verifikasi:** Jalankan aplikasi Anda, klik salah satu produk dari daftar, dan pastikan halaman detail produk muncul dengan data yang benar. Jika Anda mensimulasikan endpoint yang dilindungi, pastikan permintaan menyertakan header autentikasi.

💡 Tips Belajar Tambahan

- **Error Handling Global:** Interceptor respons `axios` sangat berguna untuk menangani error autentikasi (401/403) secara global di satu tempat, daripada menanganinya di setiap komponen.
- **Refresh Token:** Untuk aplikasi yang lebih kompleks, pelajari konsep refresh token untuk memperbarui token autentikasi yang sudah expired tanpa user perlu login ulang.
- **State Management untuk Autentikasi:** Mengelola status autentikasi (token, data user) menggunakan Context API atau library state management akan mempermudah akses informasi ini di seluruh aplikasi dan mengimplementasikan Protected Routes.

🔗 Referensi Tambahan

- React Router DOM - useParams
- Axios - Interceptors
- JWT Authentication Best Practices

Hari ini kita sudah bisa menampilkan detail produk dan yang terpenting, kita sudah belajar cara mengirim header autentikasi untuk berinteraksi dengan endpoint yang dilindungi. Ini adalah skill krusial untuk membangun aplikasi fullstack yang aman. Besok, kita akan masuk ke pengantar state management dan tugas mingguan.