📆 Hari 3 (Rabu) - Protected Routes

🎯 Tujuan Pembelajaran Hari Ini

- Memahami konsep dan kebutuhan Protected Routes dalam aplikasi React.
- Mampu membuat komponen `ProtectedRoute` menggunakan React Router v6.
- Mengintegrasikan `ProtectedRoute` dengan konfigurasi routing aplikasi.
- Menguji coba akses ke rute yang dilindungi dalam berbagai skenario autentikasi.

📚 Materi Inti (2 Jam)

1. Konsep Protected Route

Di Hari 2, kita sudah belajar cara mencegah user yang sudah login mengakses halaman Login/Register. Sekarang, kita akan membahas skenario sebaliknya: bagaimana mencegah user yang *belum* login mengakses halaman-halaman yang seharusnya hanya bisa dilihat oleh user terautentikasi, seperti halaman dashboard, profil user, pengaturan akun, atau halaman checkout.

Protected Route adalah sebuah pola atau komponen dalam routing aplikasi frontend yang bertugas memeriksa status autentikasi user sebelum mengizinkan akses ke rute tertentu. Jika user sudah terautentikasi, rute yang diminta akan ditampilkan. Jika belum, user akan diarahkan (redirect) ke halaman lain, biasanya halaman login.

Ini seperti penjaga pintu di sebuah acara eksklusif. Hanya mereka yang punya 'tiket' (token autentikasi) yang diizinkan masuk. Jika tidak punya tiket, mereka akan diminta pergi ke loket tiket (halaman login).

2. Membuat Komponen `ProtectedRoute`

Kita akan membuat komponen React baru, misalnya bernama `ProtectedRoute.jsx`, yang akan bertindak sebagai penjaga pintu ini. Komponen ini akan menggunakan logika pengecekan status autentikasi (berdasarkan keberadaan token di `localStorage` untuk saat ini) dan hook navigasi dari React Router.

Di React Router v6, pola umum untuk Protected Routes adalah dengan membuat komponen wrapper yang menggunakan hook `useOutlet` atau me-render `<Outlet />` jika user terautentikasi, dan me-render `<Navigate />` jika tidak.

```
// src/components/ProtectedRoute.jsx
import React from 'react';
import { Navigate, Outlet } from 'react-router-dom';

const ProtectedRoute = () => {
  // Ambil status autentikasi. Untuk saat ini, cek token di localStorage.
  // Di Hari 4, kita akan ganti ini dengan state dari Context API.
  const isAuthenticated = localStorage.getItem('authToken') !== null;

  // Jika user terautentikasi, render child routes (menggunakan <Outlet />)
  // Jika tidak, redirect ke halaman login
```

```
  return isAuthenticated ? <Outlet /> : <Navigate to="/login" replace />;
};

export default ProtectedRoute;
```

Penjelasan:

- Komponen `ProtectedRoute` tidak me-render elemen UI secara langsung, melainkan bertindak sebagai wrapper.
- `localStorage.getItem('authToken') !== null` adalah cara sederhana kita mengecek status login saat ini.
- `<Outlet />` adalah komponen dari `react-router-dom` yang me-render child route yang cocok. Jika `ProtectedRoute` digunakan sebagai elemen dari `<Route>` yang memiliki child `<Route>`, `<Outlet />` akan me-render komponen dari child route tersebut.
- `<Navigate to="/login" replace />` adalah komponen dari `react-router-dom` yang berfungsi seperti redirect. Ketika komponen ini di-render, React Router akan otomatis berpindah ke path `/login`. Atribut `replace` memastikan bahwa halaman saat ini (yang mencoba diakses) diganti di history browser, sehingga user tidak bisa kembali ke halaman terproteksi dengan tombol 'back'.

3. Integrasi dengan React Router v6 (`<Outlet />`)

Setelah komponen `ProtectedRoute` dibuat, kita perlu menggunakannya dalam konfigurasi routing kita. Kita akan menempatkan rute-rute yang ingin dilindungi sebagai child dari `<Route>` yang menggunakan `ProtectedRoute` sebagai elemennya.

Misalnya, jika Anda memiliki rute `/dashboard` dan `/profile` yang hanya bisa diakses setelah login, konfigurasi routing Anda di `App.js` (atau file routing utama) akan terlihat seperti ini:

```
// src/App.js (atau file routing utama Anda)
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import HomePage from './pages/HomePage';
import LoginPage from './pages/LoginPage';
import RegisterPage from './pages/RegisterPage';
import DashboardPage from './pages/DashboardPage';
import ProfilePage from './pages/ProfilePage'; // Asumsikan ada halaman
Profile
import ProtectedRoute from './components/ProtectedRoute';

function App() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<HomePage />} />
        <Route path="/login" element={<LoginPage />} />
        <Route path="/register" element={<RegisterPage />} />

        {/* Rute yang Dilindungi */}
        <Route element={<ProtectedRoute />}> {/* Gunakan ProtectedRoute
sebagai elemen wrapper */}
          <Route path="/dashboard" element={<DashboardPage />} /> {/* Child
```

```
route */}
          <Route path="/profile" element={<ProfilePage />} />     {/* Child
route lain */}
        </Route>

        {/* Tambahkan rute lain jika ada */}
        {/* <Route path="*" element={<NotFoundPage />} /> */}

      </Routes>
    </Router>
  );
}

export default App;
```

Dalam konfigurasi ini, ketika user mencoba mengakses `/dashboard` atau `/profile`, React Router akan me-render `ProtectedRoute`. Di dalam `ProtectedRoute`, status autentikasi akan dicek. Jika user login, `<Outlet />` akan me-render `DashboardPage` atau `ProfilePage` sesuai path-nya. Jika user belum login, `ProtectedRoute` akan me-render `<Navigate to="/login" replace />`, yang akan mengarahkan user ke halaman login.

🏋️ Praktik Mandiri (8 Jam)

1. **Lanjutkan proyek e-commerce Anda.** Pastikan Anda sudah menyelesaikan praktik Hari 1 dan Hari 2 (login/logout dengan `localStorage`, redirect dari login/register jika sudah login). Pastikan juga Anda sudah memiliki komponen `DashboardPage.jsx` sederhana dan route `/dashboard`.
2. **Buat komponen `ProtectedRoute.jsx`:** Buat file baru `src/components/ProtectedRoute.jsx` dan salin kode komponen `ProtectedRoute` seperti contoh di atas. Pastikan Anda mengimpor `Navigate` dan `Outlet` dari `react-router-dom`.
3. **Modifikasi konfigurasi routing:** Di file routing utama Anda (misalnya `App.js`), impor komponen `ProtectedRoute`. Ubah konfigurasi route `/dashboard` agar berada di dalam `<Route element={<ProtectedRoute />}>` seperti contoh di atas.
4. **Uji coba akses `/dashboard` saat belum login:**
   - Pastikan Anda sudah logout (token tidak ada di `localStorage`).
   - Coba akses `/dashboard` langsung dari URL browser.
   - **Verifikasi:** Anda seharusnya langsung diarahkan ke halaman `/login`.
5. **Uji coba akses `/dashboard` setelah login:**
   - Login kembali (pastikan token tersimpan di `localStorage`).
   - Coba akses `/dashboard` langsung dari URL browser.
   - **Verifikasi:** Anda seharusnya bisa melihat konten `DashboardPage`.
6. **(Opsional) Buat halaman lain yang dilindungi:** Buat komponen halaman sederhana lain (misalnya `ProfilePage.jsx`) dan tambahkan route `/profile` di dalam `<Route element={<ProtectedRoute />}>`. Uji coba akses halaman `/profile` dalam kondisi login dan logout.

💡 Tips Belajar Tambahan

- **Struktur Folder:** Pertimbangkan untuk menyimpan komponen `ProtectedRoute` di folder `src/components` atau `src/utils`.
- **Pengecekan Status:** Saat ini kita masih menggunakan `localStorage` untuk mengecek status. Di Hari 4, kita akan beralih ke Context API untuk manajemen state autentikasi yang lebih baik dan terpusat.
- **Error Handling:** Pertimbangkan bagaimana menangani kasus jika token ada tapi sudah tidak valid di backend (misalnya, expired). Backend biasanya akan merespons dengan status 401 (Unauthorized) atau 403 (Forbidden). Anda bisa menambahkan logika di interceptor `axios` (seperti yang dibahas sekilas di Minggu 7 Hari 4) atau di Context API untuk menghapus token lokal dan me-redirect user ke login saat menerima respons 401/403.

🔗 Referensi Tambahan

- React Router Documentation - Auth Examples (Lihat contoh "Auth flow")
- React Router Documentation - Outlet
- React Router Documentation - Navigate

Hari ini kita sudah berhasil mengimplementasikan Protected Routes, sebuah pola krusial untuk mengamankan rute di aplikasi frontend. Dengan komponen `ProtectedRoute`, kita bisa dengan mudah menentukan rute mana saja yang hanya bisa diakses oleh user terautentikasi. Besok, kita akan menyempurnakan manajemen status autentikasi ini menggunakan Context API agar status login bisa diakses dan diperbarui dengan lebih efisien di seluruh aplikasi.