📆 Hari 1 (Senin) - Menyimpan & Mengelola Token Autentikasi

🎯 Tujuan Pembelajaran Hari Ini

- Memahami pentingnya token autentikasi di frontend.
- Mengenal berbagai opsi penyimpanan token di frontend (`localStorage`, `sessionStorage`, Cookies) beserta kelebihan dan kekurangannya.
- Mampu mengimplementasikan penyimpanan, pengambilan, dan penghapusan token menggunakan `localStorage`.
- Memahami alur dasar logout di sisi frontend.

📚 Materi Inti (2 Jam)

1. Review Proses Login

Sebelum masuk ke penyimpanan token, mari kita ingat kembali alur login yang sudah dibahas di Minggu 7 Hari 3:

- User memasukkan kredensial (email/password) di form login.
- Frontend mengirim kredensial ke endpoint login backend.
- Backend memverifikasi kredensial.
- Jika valid, backend menghasilkan token autentikasi (misalnya, JWT) dan mengirimkannya kembali ke frontend dalam respons.

2. Pentingnya Token di Frontend

Setelah user berhasil login dan backend mengirimkan token, token ini menjadi 'kunci' bagi user untuk mengakses sumber daya atau endpoint di backend yang membutuhkan autentikasi. Setiap kali frontend perlu berkomunikasi dengan endpoint yang dilindungi (misalnya, mengambil data profil user, membuat pesanan), token ini harus disertakan dalam permintaan agar backend tahu siapa user yang melakukan permintaan tersebut dan apakah user tersebut berhak mengakses sumber daya tersebut.

3. Opsi Penyimpanan Token di Frontend: `localStorage`, `sessionStorage`, Cookies

Ada beberapa cara untuk menyimpan token autentikasi di sisi browser (frontend). Setiap metode memiliki karakteristik dan implikasi keamanan yang berbeda:

- **`localStorage`:**
    - **Cara Kerja:** Menyimpan data sebagai string key-value pair di browser. Data ini persisten, artinya tetap ada bahkan setelah browser ditutup dan dibuka kembali.
    - **Kelebihan:** Mudah digunakan, data persisten.
    - **Kekurangan:** Rentan terhadap serangan XSS (Cross-Site Scripting). Jika ada script berbahaya yang berhasil dieksekusi di halaman Anda, script tersebut bisa mengakses dan mencuri token dari `localStorage`. Token yang disimpan di `localStorage` tidak otomatis dikirim dengan

setiap request HTTP; Anda harus mengambilnya secara manual dan menambahkannya ke header request.

- `sessionStorage`:
  - **Cara Kerja:** Mirip dengan `localStorage`, menyimpan data sebagai string key-value pair. Namun, data di `sessionStorage` hanya bertahan selama sesi browser (tab atau jendela) aktif. Data akan hilang saat tab atau jendela ditutup.
  - **Kelebihan:** Data tidak persisten setelah sesi berakhir, sedikit lebih aman dari `localStorage` dalam skenario tertentu (meskipun masih rentan XSS dalam sesi yang sama).
  - **Kekurangan:** Rentan XSS dalam sesi yang sama, tidak otomatis dikirim dengan setiap request.
- **Cookies:**
  - **Cara Kerja:** Data disimpan di browser dan secara otomatis dikirim dengan setiap request HTTP ke domain yang sama (jika diatur dengan benar).
  - **Kelebihan:** Bisa diatur flag `HttpOnly` (mencegah akses dari JavaScript, mengurangi risiko XSS) dan `Secure` (hanya dikirim melalui HTTPS). Otomatis dikirim dengan request.
  - **Kekurangan:** Rentan terhadap serangan CSRF (Cross-Site Request Forgery) jika tidak diatur dengan benar (misalnya, menggunakan `SameSite` attribute). Ukuran penyimpanan terbatas.

4. Memilih `localStorage` untuk Latihan

Untuk tujuan pembelajaran dan kemudahan implementasi awal, kita akan fokus menggunakan `localStorage`. Penting untuk diingat bahwa di aplikasi produksi yang sensitif terhadap keamanan, penggunaan `localStorage` untuk menyimpan token autentikasi jangka panjang (seperti JWT) tidak disarankan karena risiko XSS. Opsi yang lebih aman seringkali melibatkan kombinasi cookie `HttpOnly` untuk refresh token dan menyimpan access token di memori (state management) atau menggunakan pola lain yang lebih kompleks.

5. Implementasi Penyimpanan Token

Setelah backend merespons sukses pada endpoint login dan mengirimkan token, Anda perlu mengambil token tersebut dari respons dan menyimpannya di `localStorage`. Ini biasanya dilakukan di dalam handler `onSubmit` pada komponen login Anda.

```
// Contoh di dalam handler submit form login
const handleSubmit = async (e) => {
  e.preventDefault();
  try {
    const response = await axios.post('/api/auth/login', { email, password
});
    const token = response.data.token; // Asumsi token ada di
response.data.token

    // Menyimpan token di localStorage
    localStorage.setItem('authToken', token); // Gunakan key yang
konsisten, misalnya 'authToken'

    // Redirect user ke halaman lain (misalnya dashboard)
```

```
    navigate('/dashboard');

  } catch (err) {
    console.error('Login failed:', err);
    // Tampilkan pesan error ke user
  }
};
```

6. Mengambil Token

Token yang tersimpan di `localStorage` perlu diambil saat Anda ingin melakukan permintaan ke endpoint yang dilindungi. Anda akan mengambil token ini dan menambahkannya ke header `Authorization` pada request `axios`.

```
// Contoh mengambil token sebelum request API
async function fetchDataProtected() {
  const token = localStorage.getItem('authToken'); // Ambil token dari
localStorage

  if (!token) {
    console.log('User not authenticated. Redirecting to login.');
    // Lakukan redirect ke halaman login jika token tidak ada
    // navigate('/login'); // Jika di dalam komponen React
    return;
  }

  try {
    const response = await axios.get('/api/protected-resource', {
      headers: {
        Authorization: `Bearer ${token}` // Menambahkan header
Authorization
      }
    });
    console.log('Protected data:', response.data);
  } catch (err) {
    console.error('Error fetching protected data:', err);
    // Tangani error, misalnya token expired (status 401/403)
    if (err.response && (err.response.status === 401 || err.response.status
=== 403)) {
      console.log('Token invalid or expired. Please login again.');
      localStorage.removeItem('authToken'); // Hapus token yang tidak valid
      // navigate('/login'); // Redirect ke halaman login
    }
  }
}
```

7. Menghapus Token (Logout)

Saat user logout, Anda perlu menghapus token dari `localStorage` agar user tidak lagi dianggap terautentikasi. Setelah menghapus token, user biasanya diarahkan kembali ke halaman login atau halaman

utama.

```
// Contoh fungsi logout
const handleLogout = () => {
  localStorage.removeItem('authToken'); // Hapus token dari localStorage
  console.log('User logged out.');
  // Redirect user ke halaman login
  // navigate('/login'); // Jika di dalam komponen React
};
```

🏋️ Praktik Mandiri (8 Jam)

1. **Lanjutkan proyek e-commerce dari minggu 7.** Pastikan Anda memiliki halaman Login yang berfungsi (meskipun masih simulasi atau menggunakan dummy API seperti `reqres.in`).
2. **Modifikasi handler login:** Di komponen `LoginPage.jsx`, setelah simulasi atau panggilan API login berhasil, ambil 'token' dari respons (jika API dummy menyediakannya, atau simulasikan token string) dan simpan di `localStorage` menggunakan `localStorage.setItem('authToken', tokenValue);`. Setelah menyimpan token, gunakan `useNavigate` untuk mengarahkan user ke halaman lain (misalnya halaman utama `/` atau halaman dashboard `/dashboard` jika sudah ada).
3. **Buat tombol Logout:** Tambahkan tombol "Logout" di suatu tempat yang mudah diakses saat user sudah login (misalnya di header atau halaman dashboard). Buat handler untuk tombol ini.
4. **Implementasikan fungsi logout:** Di dalam handler tombol Logout, panggil `localStorage.removeItem('authToken');` untuk menghapus token. Setelah token dihapus, gunakan `useNavigate` untuk mengarahkan user kembali ke halaman `/login`.
5. **Verifikasi:** Buka browser developer tools (biasanya F12), pergi ke tab "Application", lalu pilih "Local Storage". Setelah login, pastikan ada item baru dengan key `authToken` dan value token Anda. Setelah logout, pastikan item tersebut hilang.

💡 Tips Belajar Tambahan

- **Perhatikan Keamanan:** Meskipun kita menggunakan `localStorage` untuk latihan, selalu ingat risikonya di aplikasi nyata. Pelajari lebih lanjut tentang cookie `HttpOnly` dan pola penyimpanan token yang lebih aman.
- **Gunakan Key yang Konsisten:** Selalu gunakan key yang sama (misalnya `authToken`) saat menyimpan, mengambil, atau menghapus token dari `localStorage`.
- **Error Handling:** Pertimbangkan apa yang terjadi jika `localStorage` tidak tersedia (misalnya, di browser lama atau pengaturan privasi ketat). Meskipun jarang, ini bisa terjadi.

🔗 Referensi Tambahan

- MDN Web Docs - Using the Web Storage API
- MDN Web Docs - HTTP Cookies
- JWT Introduction
- Axios Documentation - Request Config (Headers)

Hari ini kita sudah meletakkan dasar penting dalam mengelola status autentikasi di frontend dengan menyimpan token. Memahami cara menyimpan dan mengambil token adalah langkah pertama sebelum kita bisa menggunakannya untuk melindungi rute dan data di aplikasi kita. Besok, kita akan menggunakan token ini untuk mengontrol navigasi dan akses halaman.