**1. Struktur Folder dan Routing**

- **Routing Berbasis Folder: App Router menggunakan sistem routing yang intuitif berbasis folder di dalam direktori app.**
    - **Halaman Utama: Didefinisikan oleh file app/page.tsx.**
    - **Halaman Baru: Untuk membuat rute baru (misalnya /posts), cukup buat folder baru bernama posts di dalam app, lalu buat file page.tsx di dalamnya.**
- **Navigasi:**
    - **Tag <a> (Tidak Disarankan): Menggunakan tag <a> standar untuk link internal akan menyebabkan *full page reload*, yang menghapus state aplikasi dan tidak efisien.**
    - **Komponen <Link> (Disarankan): Next.js menyediakan komponen <Link> dari next/link. Komponen ini melakukan navigasi di sisi klien (*client-side navigation*), artinya hanya komponen yang berubah yang akan di-render ulang, membuat transisi antar halaman menjadi sangat cepat dan mulus.**

**2. Styling dengan CSS dan Tailwind CSS**

- **Global CSS: File app/globals.css digunakan untuk mendefinisikan gaya yang berlaku di seluruh aplikasi.**
- **CSS Modules: Untuk styling yang terisolasi pada komponen tertentu, digunakan CSS Modules. Caranya adalah dengan membuat file CSS dengan ekstensi .module.css (misal: Card.module.css). Gaya dari file ini diimpor sebagai objek dan digunakan pada className untuk mencegah konflik nama kelas CSS.**
- **Tailwind CSS: Karena sudah diinstal, *utility classes* dari Tailwind CSS bisa langsung ditulis di dalam className pada elemen JSX, mempercepat proses styling.**

**3. Komponen: Server vs. Klien**

**Ini adalah salah satu konsep paling fundamental di Next.js 13.**

- **Server Components (Default):**
    - **Semua komponen di dalam direktori app secara default adalah Server Components.**
    - **Komponen ini di-render di server, sehingga tidak bisa menggunakan *hooks* (seperti useState, useEffect) atau menangani interaksi pengguna (seperti onClick).**
    - **Keuntungannya adalah performa yang lebih baik karena kode JavaScript untuk komponen ini tidak dikirim ke browser.**
- **Client Components:**
    - **Untuk membuat komponen yang interaktif, Anda harus mengubahnya menjadi Client Component.**

    - **Caranya adalah dengan menambahkan direktif "use client"; di baris paling atas file komponen.**
    - **Komponen ini di-render di browser dan dapat menggunakan *hooks* serta menangani *event listeners*.**
- **Hybrid Components: menunjukkan cara menggabungkan keduanya. Misalnya, sebuah halaman (Server Component) dapat mengimpor dan menampilkan sebuah tombol (Client Component).**

**4. Fetching Data (Pengambilan Data)**

- **Menggunakan fetch: Data diambil dari API menggunakan fungsi fetch standar JavaScript di dalam komponen *async*.**
- **Type Safety: Interface TypeScript (interface IPost { ... }) dibuat untuk mendefinisikan struktur data yang diterima dari API. Ini membantu mencegah bug terkait tipe data.**
- **Menampilkan Data: Setelah data berhasil diambil, metode .map() digunakan untuk melakukan iterasi pada array data dan merender setiap item ke dalam komponen (misalnya, CardList).**

**5. Caching dan Rendering (Statis vs. Dinamis)**

- **Static Rendering (Default): Secara default, Next.js akan mengambil data pada saat *build time* (saat npm run build dijalankan) dan membuat halaman HTML statis. Ini membuat halaman sangat cepat, tetapi datanya tidak akan ter-update sampai aplikasi di-build ulang.**
- **Dynamic Rendering: Untuk memastikan data selalu yang terbaru, ada dua cara:**
    1. **cache: 'no-store': Menambahkan opsi ini pada fungsi fetch akan membuat halaman di-render secara dinamis di server pada setiap permintaan (*server-side rendering at runtime*).**
    2. **revalidate: Opsi ini memungkinkan *Incremental Static Regeneration* (ISR). Anda bisa menentukan durasi (dalam detik) kapan Next.js harus mengambil ulang data di latar belakang (misalnya, revalidate: 3600 untuk setiap jam).**

**6. Metode fetch**

Metode **fetch** adalah sebuah fungsi bawaan JavaScript modern yang digunakan untuk membuat permintaan jaringan (seperti mengambil data dari sebuah API). Ini adalah pengganti yang lebih kuat dan fleksibel untuk XMLHttpRequest.

- **Cara Kerja**: fetch bekerja secara *asynchronous* (tidak memblokir eksekusi kode lain) dan mengembalikan sebuah **Promise**. Promise ini akan menghasilkan sebuah objek Response yang berisi informasi tentang respons dari server.
- **Mengambil Data**: Untuk mendapatkan data sebenarnya (biasanya dalam format JSON), Anda perlu memanggil metode .json() pada objek Response tersebut, yang juga mengembalikan sebuah Promise.

**Contoh Penggunaan (async/await)**:

Ini adalah cara paling umum dan mudah dibaca untuk menggunakan fetch.

JavaScript

```
// Fungsi untuk mengambil data postingan dari API
async function ambilDataPostingan() {
  try {
    // 1. Meminta data ke URL API
    const respons = await fetch('https://jsonplaceholder.typicode.com/posts/1');

    // Cek jika permintaan tidak berhasil (misal: error 404 atau 500)
    if (!respons.ok) {
      throw new Error(`Terjadi error: ${respons.status}`);
    }

    // 2. Mengubah respons menjadi format JSON
    const data = await respons.json();

    // 3. Menampilkan data ke konsol
    console.log(data);

  } catch (error) {
    // Menangani jika ada kesalahan jaringan atau lainnya
    console.error("Tidak bisa mengambil data:", error);
  }
}

// Memanggil fungsi untuk menjalankannya
ambilDataPostingan();
```

Pada contoh di atas, await digunakan untuk menunggu hingga proses fetch dan .json() selesai sebelum melanjutkan ke baris kode berikutnya.

---

**7. Interface / Type**

**Interface** adalah fitur dari **TypeScript** (bukan JavaScript murni). Fungsinya adalah untuk mendefinisikan "kontrak" atau "bentuk" dari sebuah objek. Dengan kata lain, interface menentukan properti apa saja yang harus dimiliki sebuah objek beserta tipe datanya.

- **Tujuan**:
    - **Type Safety**: Memastikan objek yang kita gunakan sesuai dengan struktur yang diharapkan, sehingga mengurangi bug.
    - **Keterbacaan Kode**: Membuat kode lebih mudah dimengerti karena struktur datanya didefinisikan dengan jelas.
    - **Autocomplete**: Memberikan saran *autocomplete* yang akurat di editor kode.

**Contoh Penggunaan**:

Mari kita buat interface untuk data postingan yang kita ambil dengan fetch di atas. Data dari JSONPlaceholder untuk satu postingan memiliki userId, id, title, dan body.

TypeScript

```
// 1. Mendefinisikan bentuk objek Postingan dengan interface
interface IPost {
  userId: number;
  id: number;
  title: string;
  body: string;
}

// 2. Menggunakan interface untuk memberikan tipe pada variabel
async function ambilDataPostingan(): Promise<IPost> { // Fungsi ini akan mengembalikan data dengan tipe IPost
  const respons = await fetch('https://jsonplaceholder.typicode.com/posts/1');
  if (!respons.ok) {
    throw new Error("Gagal fetch data");
  }
  const data: IPost = await respons.json(); // Memastikan data yang diterima sesuai dengan IPost

  console.log(data.title); // Editor akan tahu bahwa 'data' punya properti 'title'
  return data;
```

}

Jika mencoba mengakses properti yang tidak ada di interface (misalnya data.author), TypeScript akan memberikan peringatan error.

---

**8. Metode .map()**

Metode **.map()** adalah fungsi standar yang ada pada semua **Array** di JavaScript. Fungsi ini digunakan untuk melakukan iterasi (perulangan) pada setiap elemen di dalam sebuah array dan membuat **array baru** dari hasil operasi tersebut.

- **Penting**: .map() tidak mengubah array asli, melainkan menghasilkan array yang baru.
- **Penggunaan Umum**: Di dalam pengembangan web (khususnya dengan React atau Next.js), .map() sangat sering dipakai untuk mengubah array data (misalnya dari API) menjadi daftar elemen HTML/JSX untuk ditampilkan di layar.

**Contoh Penggunaan**:

Misalkan kita mengambil 10 data postingan dan ingin menampilkannya sebagai sebuah daftar judul.

JavaScript

```
// Data array (misalnya hasil dari fetch)
const daftarPostingan = [
 { id: 1, title: 'Judul Postingan Pertama' },
 { id: 2, title: 'Judul Postingan Kedua' },
 { id: 3, title: 'Judul Postingan Ketiga' }
];

// 1. Menggunakan .map() untuk mengubah array objek menjadi array string (judul)
const daftarJudul = daftarPostingan.map(postingan => {
  return postingan.title;
});

console.log(daftarJudul);
// Output: ['Judul Postingan Pertama', 'Judul Postingan Kedua', 'Judul Postingan Ketiga']

// 2. Contoh di React/Next.js untuk membuat daftar list <li>
```

```
// function DaftarPostinganComponent() {
//   return (
//     <ul>
//       {daftarPostingan.map(postingan => (
//         <li key={postingan.id}>
//           {postingan.title}
//         </li>
//       ))}
//     </ul>
//   );
// }
```

Pada contoh kedua, .map() mengubah setiap objek postingan dalam array daftarPostingan menjadi sebuah elemen <li>. Atribut key sangat penting untuk membantu React mengidentifikasi setiap elemen secara unik saat ada perubahan data.