# Materi 4: Konsep *State* dalam React

# Silabus

| Pertemuan | Materi |
|---|---|
| 1 | JavaScript, Node.js, `npm`, dan `pnpm` |
| 2 | React, Vite, Komponen, JSX, dan Meneruskan *Props* |
| 3 | *Styling*, *Conditional & List Rendering*, dan Menanggapi *Event* |
| 4 | Konsep *State* dalam React |
| 5 | Memperbarui Nilai dalam *State* |
| 6 | *Sharing* Data Antarkomponen Menggunakan Props |
| 7 | *Routing* Menggunakan React Router |
| 8 | *Deployment* (Vercel) |
| **9** | **UTS** |
| 10 | Tailwind CSS |
| 11 | Memperdalam event dan `useState()` (memperbarui objek dan array) |
| 12 | `useReducer()` dan `useContext()` |
| 13 | `useRef()` dan `useEffect()` |
| 14 | REST API (Express), CRUD, dan deployment (Vercel) |
| 15 | Basis data (Supabase, PostgreSQL) dan `.env` |
| 16 | Otentikasi (JWT, cookies, dan hash kata sandi) |
| 17 | TypeScript |
| **18** | **Projekan** |

# Daftar isi


- **Materi 4: Konsep *State* dalam React**
- Silabus
- Daftar isi
- *State*: Memori Komponen
  - Kekurangan variabel biasa
  - Menambahkan variabel *state*
  - Hook pertama kita: `useState`
  - Anatomi dari `useState`
  - Memberikan komponen beberapa variabel *state*
  - *State* bersifat terisolasi dan *private*
  - Rangkuman

# *State*: Memori Komponen

Komponen seringkali perlu mengubah apa yang ada di layar sebagai hasil interaksi. Mengetik dalam *form* akan memperbarui nilai bidang input, mengklik "berikutnya" pada *slide* gambar akan mengubah gambar mana yang ditampilkan, mengklik "beli" akan memasukkan produk ke dalam keranjang belanja. Komponen perlu "mengingat" hal-hal tertentu: nilai input saat ini, gambar saat ini, keranjang belanja. Di React, jenis memori khusus komponen ini disebut *state*.

Kita akan belajar:

- Cara menambahkan variabel *state* dengan Hook `useState`
- Pasangan nilai apa yang dikembalikan oleh Hook `useState`
- Cara menambahkan lebih dari satu variabel *state*
- Mengapa *state* disebut lokal

## Kekurangan variabel biasa

Berikut contoh komponen yang merender nama planet. Mengklik tombol "Berikutnya" akan menampilkan planet berikutnya dengan mengubah `index` ke `1`, lalu `2`, dan seterusnya. Namun, ini **tidak akan berhasil** (silakan dicoba!):

```
const planets = ["Mercury", "Venus", "Earth"];

export default function SolarSystem() {
  let index = 0;

  function handleClick() {
    index++;
  }

  return (
    <>
      <h1>{planets[index]}</h1>
```

```
      <button onClick={handleClick}>Berikutnya</button>
    </>
  );
}
```

## Menambahkan variabel *state*

Untuk menambahkan variabel *state*, impor `useState` dari React di bagian atas file:

```
import { useState } from "react";
```

Kemudian, ganti baris ini:

```
let index = 0;
```

dengan

```
const [index, setIndex] = useState(0);
```

`index` adalah variabel *state* dan `setIndex` merupakan fungsi *setter* (fungsi untuk mengubah nilai *state*).

> Sintaks `[` dan `]` di atas disebut *array destructuring* yang memungkinkan kita untuk mengeluarkan nilai-nilai dari *array*. *Array* yang dikembalikan oleh `useState` selalu memiliki dua item.

Beginilah cara menggunakannya dengan `handleClick`:

```
function handleClick() {
  setIndex(index + 1);
}
```

Sekarang mengklik tombol "Berikutnya" akan berhasil mengganti planet saat ini:

```
const planets = ["Mercury", "Venus", "Earth"];

export default function SolarSystem() {
  const [index, setIndex] = useState(0);

  function handleClick() {
    setIndex(index + 1);
  }

  return (
    <>
      <h1>{planets[index]}</h1>
      <button onClick={handleClick}>Berikutnya</button>
    </>
  );
}
```

## Hook pertama kita: `useState`

Dalam React, `useState`, serta fungsi-fungsi lainnya yang diawali dengan ”use”, disebut Hook.

Hook adalah fungsi-fungsi khusus yang hanya tersedia saat React merender (yang akan kita bahas lebih detail di materi berikutnya). Mereka dinamai "hook" (pengait) karena mengaitkan/menghubungkan kita ke berbagai fitur React.

*State* hanyalah salah satu dari fitur tersebut, tetapi kita akan bertemu dengan Hook lainnya nanti.

**Peringatan**

Hook (fungsi-fungsi yang diawali dengan *use*) hanya dapat dipanggil di bagian atas komponen atau di Hook buatan kita sendiri (akan dipelajari di materi berikutnya). Kita tidak dapat memanggil Hook di dalam *decision* (kondisi), *loop* (perulangan), atau fungsi lain di dalam komponen (fungsi bersarang). Meskipun Hook hanyalah fungsi, tapi ia kita

anggap sebagai deklarasi untuk kebutuhan komponen kita, seperti saat kita meng-“import” modul di bagian atas file.

## Anatomi dari `useState`

Saat kita memanggil `useState`, kita memberi tahu React bahwa kita ingin komponen ini mengingat sesuatu:

```
const [index, setIndex] = useState(0);
```

Dalam contoh di atas, kita ingin React mengingat `index`.

Konvensi yang disarankan untuk menamai sepasang nilai tersebut adalah `const [sesuatu, useSesuatu]`. Meskipun kita dapat menamainya dengan apa pun yang kita suka, tetapi konvensi tersebut akan membuat segalanya lebih mudah dipahami di seluruh *project* yang dikerjakan.

Satu-satunya argumen `useState` adalah nilai awal dari variabel *state* kita. Dalam contoh ini, nilai awal `index` diatur ke `0` dengan `useState(0)`.

Setiap kali komponen kita merender, `useState` memberi kita *array* yang berisi dua nilai:

1. Variabel *state* (`index`) berisi nilai yang kita simpan.
2. Fungsi *setter state* (`setIndex`) yang dapat memperbarui variabel *state* dan memicu React untuk merender komponen lagi.

Inilah cara kerja `useState`:

```
const [index, setIndex] = useState(0);
```

1. **Komponen merender pertama kali**. Karena kita meneruskan `0` ke `useState` sebagai nilai awal untuk `index`, maka akan mengembalikan `[0, setIndex]`. React mengingat bahwa `0` adalah nilai *state* terbaru saat ini.
2. **Pengguna memperbarui *state***. Ketika pengguna mengklik tombol, itu akan memanggil `setIndex(index + 1)`. `index` saat ini adalah `0`, jadi itu sama seperti

`setIndex(1)`. Ini memberitahu React untuk mengingat bahwa `index` sekarang adalah `1` dan memicu render lagi.

3. **Render kedua**. React masih membaca `useState(0)`, tetapi karena React ingat bahwa kita telah mengatur `index` ke `1`, `useState` akan mengembalikan `[1, setIndex]`.
4. Dan seterusnya!

## Memberikan komponen beberapa variabel *state*

Kita dapat memiliki banyak variabel *state* sebanyak yang kita inginkan dalam satu komponen. Komponen ini memiliki dua variabel status, sebuah angka `index` dan sebuah *boolean* `showMore` yang diaktifkan saat kita mengklik "Tampilkan detail":

```
const planets = ["Mercury", "Venus", "Earth"];

export default function SolarSystem() {
  const [index, setIndex] = useState(0);

  function handleClick() {
    setIndex(index + 1);
  }

  return (
    <>
      <h1>{planets[index]}</h1>
      {showMore && <p>Planet ke-{index + 1} dalam tata surya.</p>}
      <button onClick={handleMoreClick}>
        {showMore ? "Sembunyikan" : "Tampilkan"} detail
      </button>
      <button onClick={handleClick}>Berikutnya</button>
    </>
  );
}
```

Memiliki beberapa variabel *state* adalah cara yang baik jika *state*-nya tidak saling terkait, seperti `index` dan `showMore` dalam contoh di atas. Tetapi jika kita menemukan bahwa kita

sering mengubah dua variabel status secara bersamaan, mungkin akan lebih mudah untuk menggabungkannya menjadi satu. Misalnya, jika kita memiliki *form* dengan banyak bidang, akan lebih mudah untuk memiliki satu variabel *state* yang menyimpan objek daripada variabel *state* untuk menyimpan nilai per bidang. Baca dokumentasi React: Choosing the State Structure untuk tips lainnya.

## *State* bersifat terisolasi dan *private*

*State* bersifat lokal untuk *instance* komponen yang dirender di layar. Dengan kata lain, jika kita merender komponen yang sama dua kali, setiap salinan akan memiliki *state* yang benar-benar terisolasi! Mengubah salah satunya tidak akan mempengaruhi yang lain.

Dalam contoh ini, komponen `SolarSystem` dari contoh sebelumnya dirender dua kali tanpa perubahan pada logikanya. Coba klik tombol di dalam setiap komponen `SolarSystem`. Perhatikan bahwa *state* mereka independen:

```
import SolarSystem from "./SolarSystem.js";

export default function Page() {
  return (
    <div className="Page">
      <SolarSystem />
      <SolarSystem />
    </div>
  );
}
```

Inilah yang membuat *state* berbeda dari variabel biasa yang biasa kita deklarasikan di bagian atas *module* kita. *State* tidak terikat pada pemanggilan fungsi tertentu atau tempat dalam kode, tetapi bersifat "lokal" ke tempat tertentu di layar. Kita merender dua komponen `<SolarSystem />`, sehingga *state*-nya disimpan secara terpisah.

Perhatikan juga bagaimana komponen `Page` tidak "tahu" apapun tentang *state* `SolarSystem` atau bahkan apakah *state* tersebut ada. Tidak seperti *props*, *state* sepenuhnya *private* untuk setiap komponen yang mendeklarasikannya. Komponen induk tidak dapat mengubahnya. Ini memungkinkan kita menambahkan *state* ke komponen apa pun atau menghapusnya tanpa memengaruhi komponen lainnya.

Bagaimana jika kita ingin kedua komponen `SolarSystem` menjaga *state*-nya tetap sinkron? Cara yang tepat untuk melakukannya di React adalah menghapus *state* dari komponen anak dan menambahkannya ke *ancestor* yang sama terdekat. Topik ini dapat dipelajari di dokumentasi React: Sharing State Between Components.

## Rangkuman

- Gunakan variabel *state* saat komponen perlu "mengingat" beberapa informasi di antara render (agar informasi itu tidak hilang saat komponen dirender ulang).
- Variabel *state* dideklarasikan dengan memanggil Hook `useState`.
- Hook adalah fungsi-fungsi khusus yang dimulai dengan `use`. Mereka meng-"hook" (mengaitkan) kita ke fitur React seperti *state*.
- Hook mungkin mengingatkan kita tentang pengimporan: mereka harus dipanggil tanpa kondisi (di luar *decision* dan *loop*). Memanggil Hook, termasuk `useState`, hanya valid pada level teratas dari sebuah komponen atau Hook lainnya.
- Hook `useState` mengembalikan sepasang nilai: *state* saat ini dan fungsi untuk memperbaruinya.
- Kita dapat memiliki lebih dari satu variabel *state* (meskipun tidak ada “identifier”, React dapat mencocokkan kembali masing-masing variabel pada setiap render berdasarkan urutan deklarasinya).
- *State* bersifat *private* untuk setiap komponen. Jika kita merendernya di dua tempat, setiap salinan mendapatkan *state*-nya sendiri.