Front End Bootcamp
PT. Mahardika Solusi Teknologi

Ariel Arliyanus, S.Kom

Chapter 14 – React JS State & Component Lifecycle

2021 © All Right Reserved

Dilarang memperbanyak dan/atau meng-copy sebagian atau seluruhmaterial dalam dokumen ini tanpa persetujuan tertulis dari PT. Mahardika Solusi Teknologi

Chapter 14

React JS State & Component Lifecycle

Pada materi sebelumnya kita sudah mempelajari tentang components dan props, tapi pada materi tersebut pengolahan data di sana masih statis atau tidak berubah-rubah, pada materi ini kita akan mempelajari bagaiana merubah data dengan dinamis

State

State mirip dengan *props*, tetapi bersifat pribadi dan sepenuhnya dikendalikan oleh component. misal kita punya state angka 1 di suatu *component*, itu brarti state angka 1 itu hanya di miliki oleh component tersebut. lalu perubahan state tersebut erat hubungannya dengan *components lifecycle*

Components lifecycle

Masing-masing Component memiliki beberapa "*lifecycle method*" yang bisa ditimpa untuk menjalankan kode pada waktu tertentu dalam proses.

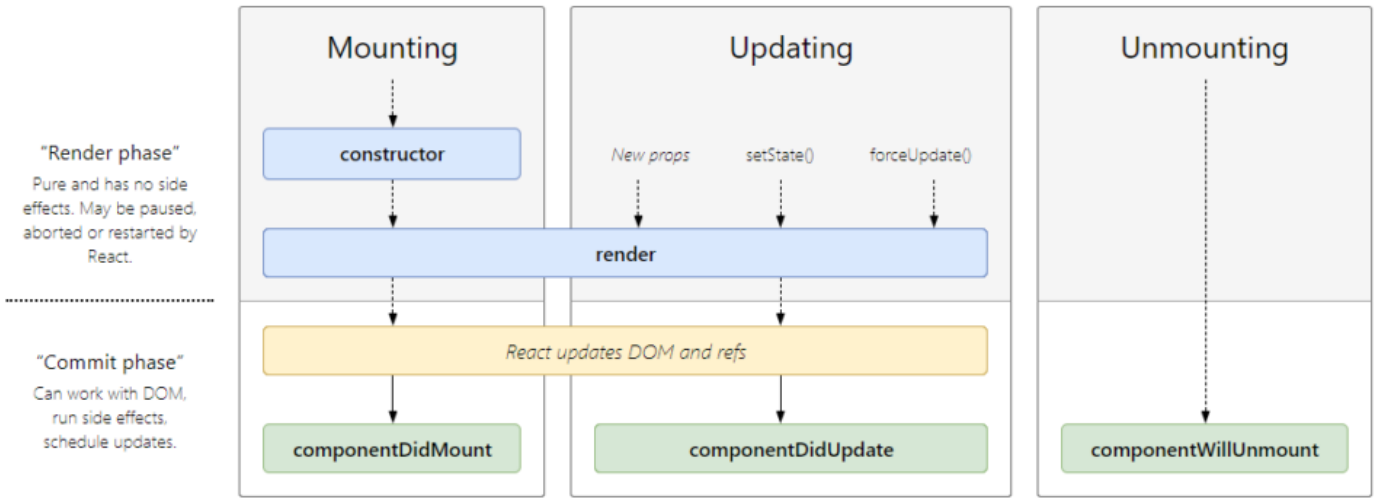

Diagram ini di sadur dari https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/
menurut diagram diatas. *lifecycle* method pada components terbagi kepada tiga fase yaitu:

Mounting (Pemasangan)

Fase Mounting adalah fase ketika components di buat atau pertama kali di render ke DOM, didalam fase ini terdapat method-method yang umum digunakan diantaranya:

- **constructor()**
- **render()**
- **componentDidMount()**

penjelasan method-method tersebut akan di jelaskan nanti satu persatu.

Updating
Fase updating adalah fase ketika sebuah component akan di render ulang, biasanya ini terjadi ketika ada perubahan pada state atau props yang mengakibatkan perubahan DOM. didalam fase ini terdapat method-method yang digunakan diantaranya:

- **render()**
- **componentDidUpdate()**

penjelasan method-method tersebut akan di jelaskan nanti satu persatu

Unmounting (Pelepasan)
Fase unmounting adalah fase ketika component di hapus dari DOM. Pada fase ini hanya ada satu method yang akan di eksekusi yaitu **componentWillUnmount()**, yang di jalankan sebelum sebuah component di hapus dari DOM

Penjelasan Method-Method lifecycle

berikut ini penjelasan mengenai method-method yang umum di gunakan dalam components lifecycle:

constructor()
constructor dalam class React Component dipanggil sebelum dipasang (mounted). Saat mengimplementasikan constructor untuk subkelas React.Component, kita harus memanggil super(props) sebelum statement lainnya. Jika tidak, this.props akan bernilai undefined did dalam constructor, yang bisa menyebabkan bugs.

pada umumnya constructor dalam React hanya digunakan untuk inisialisasi state lokal dengan ditetapkan dalam object ke this.state.

tidak boleh memanggil setState() dalam constructor(). dan sangat disarankan tidak memanggil props di dalam constructor. berikut ini contoh penerapan constructor dalam sebuah class Timer:

```
class Timer extends Component{
  constructor(props){
    super(props)
    this.state = {
      time: 0
    }
  }
}
```

render()
method render() merupakan satu-satunya metjod yang paling dibutuhkan dalam class Component.

Saat dipanggil, method ini akan memeriksa this.props dan this.state serta mengembalikan tipe berikut:

Element React. Umumnya dibuat lewat JSX. Misalnya, dan merupakan elemen React yang memerintahkan React untuk me-render sebuah simpul DOM, atau komponen yang didefinisikan pengguna.
Array dan fragment. Memungkinkan Anda untuk mengembalikan beberapa elemen sekaligus dari

render. Lihat dokumentasi tentang fragment untuk detail lebih lanjut.
Portal. Memungkinkan Anda untuk me-render anak ke subpohon DOM yang berbeda. Lihat dokumentasi tentang portals for more details.
String dan angka. Tipe ini akan di-render sebagai simpul teks dalam DOM.
Boolean atau null. Tidak me-render (umumnya ada untuk mendukung pola return test && , dengan nilai test yang bertipe boolean.)
Fungsi render() harus bersifat murni (pure), yang berarti fungsi ini tidak mengubah state komponen, mengembalikan hasil yang sama setiap kali dipanggil, dan tidak berinteraksi langsung dengan browser.

Jika Anda harus berinteraksi dengan browser, jalankan prosesnya dalam componentDidMount() atau dalam metode lifecycle lainnya saja. Dengan menjaga render() bersifat murni, cara kerja komponen lebih mudah dibayangkan.

berikut ini contoh penerapannya dalam class Timer:

```
class Timer extends Component{
  constructor(props){
    super(props)
    this.state = {
      time: 0
    }
  }

  render(){
    return(
      <>
        <h1 style={{textAlign: "center"}}>
          {this.state.time}
        </h1>
      </>
    )
  }
}
```

componentDidMount()

componentDidMount() adalah method yang di panggil setelah render namun kita bisa menginisialisasi kembali data kita di dalam method ini, dan di perbolehkan menggunakan setState di dalam method ini. dan kita dapat menggunakan props di dalam method ini

dalam penerapannya biasanya, componentDidMount() di pakai untuk mengambil data dari Rest API.

berikut ini penerapan componentDidMount() di dalam class Timer:

```
class Timer extends Component{
  componentDidMount(){
    if (this.props.start !== undefined){
      this.setState({time: this.props.start})
    }
    this.timerID = setInterval(
      () => this.tick(),
      1000
    );
  }

  tick() {
    this.setState({
      time: this.state.time + 1
    });
  }
}
```

componentDidUpdate()

componentDidUpdate() langsung dipanggil setelah terjadi perubahan. Metode ini tidak dipanggil dalam proses render awal.

Gunakan metode ini sebagai kesempatan untuk beroperasi pada DOM ketika komponen diperbarui. Metode ini juga merupakan tempat yang baik untuk menjalankan pemanggilan jaringan, selama Anda bisa membandingkan prop saat ini dengan prop sebelumnya (misalnya, permintaan jaringan mungkin tidak diperlukan jika prop tidak berubah).

```
componentDidUpdate(prevProps) {
   // Penggunaan umum (Jangan lupa untuk membandingkan props):
   if (this.props.userID !== prevProps.userID) {
     this.fetchData(this.props.userID);
   }
 }
```

Anda bisa langsung memanggil setState() dalam componentDidUpdate() tetapi perhatikan bahwa pemanggilannya harus dibungkus dalam sebuah kondisi seperti contoh di atas, atau Anda akan mengakibatkan perulangan yang tak terbatas. Ini juga akan mengakibatkan proses render ekstra yang walau tidak tampak ke pengguna, bisa berdampak pada kinerja komponen. Jika Anda mencoba “mencerminkan” beberapa state ke sebuah prop yang datang dari tingkat yang lebih tinggi, pertimbangkan untuk menggunakan secara langsung prop-nya. Baca lebih lanjut tentang mengapa menyalin props ke state bisa menyebabkan bug.

Jika komponen Anda mengimplementasikan lifecycle getSnapshotBeforeUpdate() (yang sangat jarang), nilai yang dikembalikan akan diteruskan sebagai parameter ”snapshot” ketiga ke componentDidUpdate(). Jika tidak, parameter ini akan bernilai undefined.

componentWillUnmount()

componentWillUnmount() dipanggil langsung sebelum komponen dilepas dan dihancurkan. Lakukan pembersihan yang diperlukan, misalnya menghancurkan timer, membatalkan permintaan jaringan, atau membersihkan semua langganan yang dibuat dalam componentDidMount().

Kita tidak bisa memanggil setState() dalam componentWillUnmount() karena komponen tidak akan pernah di-render ulang. Segera setelah komponen dilepas, komponen tersebut tidak akan dipasang kembali.

contoh penerapannya

berikut ini contoh penerapan method-method diatas dalam class Timer:

```
import React, {Component} from 'react'

class Timer extends Component{
  constructor(props){
    super(props)
    this.state = {
      time: 0
    }
  }

  componentDidMount(){
    if (this.props.start !== undefined){
      this.setState({time: this.props.start})
    }
    this.timerID = setInterval(
      () => this.tick(),
      1000
    );
  }

  componentWillUnmount(){
    clearInterval(this.timerID);
  }

  tick() {
    this.setState({
      time: this.state.time + 1
    });
  }

  render(){
    return(
      <>
        <h1 style={{textAlign: "center"}}>
          {this.state.time}
        </h1>
      </>
    )
  }
}
export default Timer
```

Tugas

Buatlah sebuah component yang di munculkan di app.js (tugas sebelumnya tetap dimunculkan) dengan ketentuan component memiliki dua timer yaitu jam sekarang dan hitung mundur seperti tampilan di bawah ini:

dengan ketentuan start hitung mundur mulai dari diatas atau sama dengan 100, ketika hitung mundur sudah mencapai angka 0 maka buatlah smua component di tugas pertama hilang (gunakan method-method dalam component lifecycle)