



Front End React Development sesi 12





Component Lifecycle

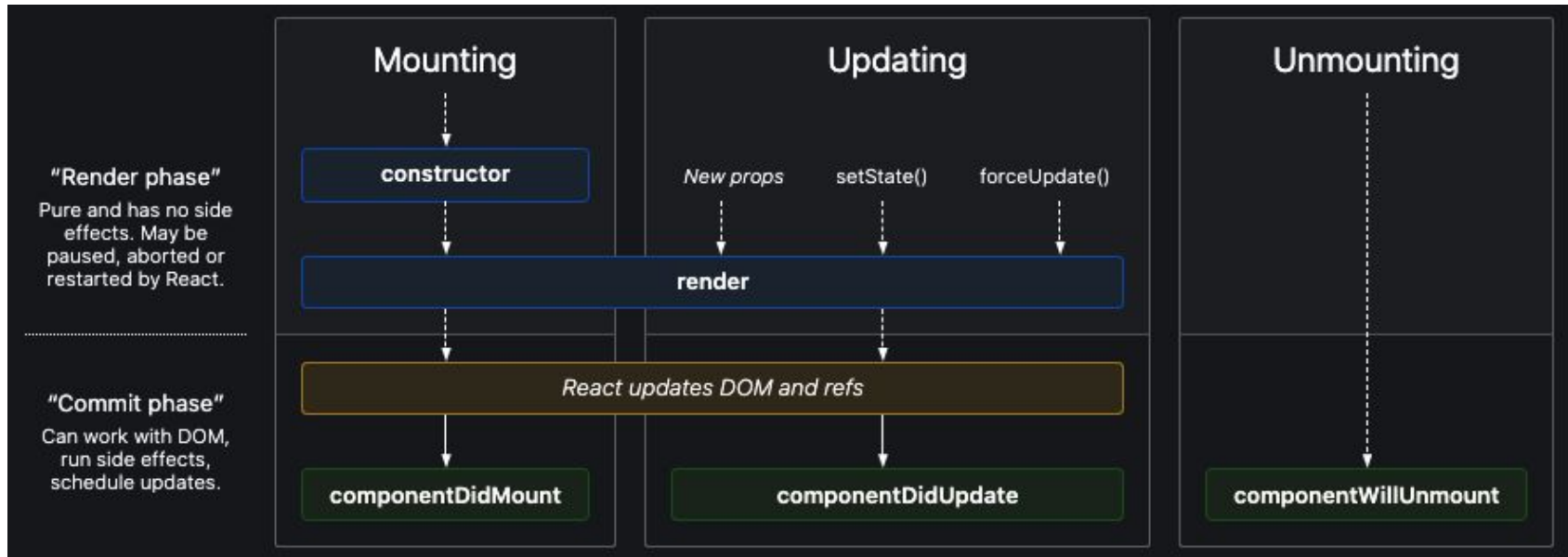
Sesi 12 | Component Lifecycle - Class Component

Analogi sederhana untuk Lifecycle adalah seperti jika kita mempunyai sebuah Restoran. Hal-hal di bawah ini bisa disebut sebagai lifecycle sebuah Restoran sehari-hari nya. Dan kita bisa memanfaatkan setiap kegiatannya untuk menyisipkan hal yang kita butuhkan



Sesi 12 | Component Lifecycle - Class Component

Pada dasarnya, Lifecycle (siklus hidup) adalah urutan kejadian atau keadaan yang terjadi secara default dan berjalan secara teratur. Lifecycle pada React, berarti urutan kejadian atau keadaan yang terjadi ketika kita meng-implementasikan React. Urutannya sendiri sudah didefinisikan secara default dalam React nya dan bisa kita manfaatkan untuk berbagai keperluan. Berikut adalah diagram common lifecycle dalam React, pada Class Component



Sesi 12 | Component Lifecycle - Class Component

MOUNTING

constructor()

Akan dipanggil saat sebuah komponen diinisialisasi. Seperti OOP dan class pada umumnya, constructor akan dipanggil pertama kali saat proses inisialisasi sebuah instance. Biasanya kita mendefinisikan state awal sebuah komponen dan melakukan deklarasi variable dan binding class method

render()

Setiap komponen class harus memiliki method ini dan harus me-return sebuah komponen. Class method ini harus bersifat pure function dan jangan ada proses perubahan state disini

componentDidMount()

Akan dipanggil ketika sebuah komponen selesai diletakkan di DOM sebuah browser. Method yang tepat untuk melakukan async request untuk mengambil data dari API. Hasil data yang didapat disimpan kedalam internal state dan akan men-trigger render()

```
1 class Clock extends React.Component {
2   constructor(props) {
3     super(props);
4     this.state = {date: new Date()};
5   }
6
7   componentDidMount() {
8     this.timerID = setInterval(
9       () => this.tick(),
10      1000
11    );
12  }
13
14  tick() {
15    this.setState({
16      date: new Date()
17    });
18  }
19
20  render() {
21    return (
22      <div>
23        <h1>Hello, world!</h1>
24        <h2>It is {this.state.date.toLocaleTimeString()}</h2>
25      </div>
26    );
27  }
28 }
29
```

Sesi 12 | Component Lifecycle - Class Component

UPDATING

shouldComponentUpdate (nextProps, nextState)

Default behaviour dari React adalah selalu re-render setiap ada perubahan state atau props. Dengan function ini, kita bisa membatasi, keadaan / kejadian apa saja yang bisa men-trigger re-render. Default return value dari function ini adalah true.

Pada contoh di samping ini, kita mau bilang bahwa kita akan melakukan re-rendering hanya jika data users nya bertambah atau berkurang

```
1 class Halu extends React.Component {
2   constructor(props) {
3     super(props);
4     this.state = {
5       count: 0,
6       users: []
7     };
8   }
9
10  shouldComponentUpdate(nextProps, nextState) {
11    if (this.state.users.length !== nextState.users.length) {
12      return true;
13    }
14
15    return false;
16  }
17 }
```



Sesi 12 | Component Lifecycle - Class Component

UPDATING

componentDidUpdate (prevProps, prevState)

Method yang satu ini akan dipanggil tepat setelah render dipanggil. Kita bisa gunakan untuk operasi DOM atau request async untuk data berikutnya disini. Prefetch misalnya

```
1 class Halu extends React.Component {
2   constructor(props) {
3     super(props);
4     this.state = {
5       isLoading: true,
6       data: []
7     };
8   }
9
10  componentDidUpdate(preProps) {
11    if (prevProps.selectedState !== this.props.selectedState) {
12      fetch('https://pathToApi.com')
13        .then(resp => resp.json())
14        .then(respJson => {
15          // do what ever you want with your `response`
16          this.setState({
17            isLoading: false,
18            data: respJson,
19          });
20        })
21        .catch(err => {
22          console.log(err)
23        })
24    }
25  }
26 }
```



Sesi 12 | Component Lifecycle - Class Component

UNMOUNTING

componentWillUnmount ()

Akan dipanggil sebelum kita menghapus komponen dari DOM. Kita bisa melakukan bersih-bersih di class method ini

```
1 class Halu extends React.Component {
2   // eventData add event listener
3   componentDidMount() {
4     eventData.addListener()
5   }
6
7   // eventData remove event listener
8   componentWillUnmount() {
9     eventData.removeListener()
10  }
```



Sesi 12 | Component Lifecycle - Class Component

BONUS : ERROR HANDLING

componentDidCatch ()

Mulai React 16, kita bisa menggunakan class method ini. Ketika terjadi kesalahan, jenis kesalahan apapun, kita sekarang bisa menampilkan pesan error yang lebih baik

```
1 class ErrorBoundary extends React.Component {
2   constructor(props) {
3     super(props);
4     this.state = { hasError: false };
5   }
6
7   static getDerivedStateFromError(error) {
8     // Update state so the next render will show the fallback UI.
9     return { hasError: true };
10  }
11
12  componentDidCatch(error, info) {
13    // Example "componentStack":
14    //   in ComponentThatThrows (created by App)
15    //   in ErrorBoundary (created by App)
16    //   in div (created by App)
17    //   in App
18    logComponentStackToMyService(info.componentStack);
19  }
20
21  render() {
22    if (this.state.hasError) {
23      // You can render any custom fallback UI
24      return <h1>Something went wrong.</h1>;
25    }
26
27    return this.props.children;
28  }
29 }
30
```

Sesi 12 | Component Lifecycle - Function Component

Kita tentunya pernah melakukan pengambilan data, berlangganan data (subscription), atau secara manual mengubah DOM dari komponen React sebelumnya. Kami menyebut operasi-operasi seperti ini “efek samping (side effects)” (atau singkatnya “efek (effects)”) karena dapat mempengaruhi komponen lain dan tidak dapat dilakukan pada saat proses render.

Effect Hook, `useEffect`, menambahkan kemampuan untuk melakukan “efek samping” dari sebuah function component. Hook ini memiliki fungsi yang sama dengan `componentDidMount`, `componentDidUpdate`, dan `componentWillUnmount` pada kelas React, tetapi disatukan menjadi satu API

Karena kita akan menggunakan Function Component untuk kedepannya, maka Hook ini lah yang akan banyak kita pakai.

```
1  import React, { useState, useEffect } from 'react';
2
3  function Example() {
4    const [count, setCount] = useState(0);
5
6    // Similar to componentDidMount and componentDidUpdate:
7    useEffect(() => {
8      // Update the document title using the browser API
9      document.title = `You clicked ${count} times`;
10   });
11
12   return (
13     <div>
14       <p>You clicked {count} times</p>
15       <button onClick={() => setCount(count + 1)}>
16         Click me
17       </button>
18     </div>
19   );
20 }
21
```



The background is a solid dark blue. It features several decorative geometric elements: a light blue triangle in the upper left, a dark blue circle in the upper right, a light blue rectangle in the top right corner, a white circle in the lower left, and a white arc in the bottom right corner.

Fetch Data⁺

Sesi 12 | Fetch Data - Penerapan Ajax Xhr

CLASS COMPONENT

Kita akan memanfaatkan **componentDidMount()** untuk melakukan data fetching dan mengisinya kepada local state

```
1  import React from 'react'
2
3  export default class Halu extends React.Component {
4    constructor(props) {
5      super(props)
6      this.state = {
7        todos: []
8      }
9    }
10
11   componentDidMount() {
12     fetch('https://jsonplaceholder.typicode.com/todos')
13       .then((response) => response.json())
14       .then((data) => this.setState({ todos: data.slice(0, 20) }))
15   }
16
17   render() {
18     return (
19       <div>
20         <table className="table">
21           <tbody>
22             {
23               this.state.todos.map((todo, index) => (
24                 <tr key={index}>
25                   <td>{todo.id}</td>
26                   <td>{todo.title}</td>
27                   <td>{todo.completed ? "v" : "x"}</td>
28                 </tr>
29               ))
30             }
31           </tbody>
32         </table>
33       </div>
34     )
35   }
36 }
37
```



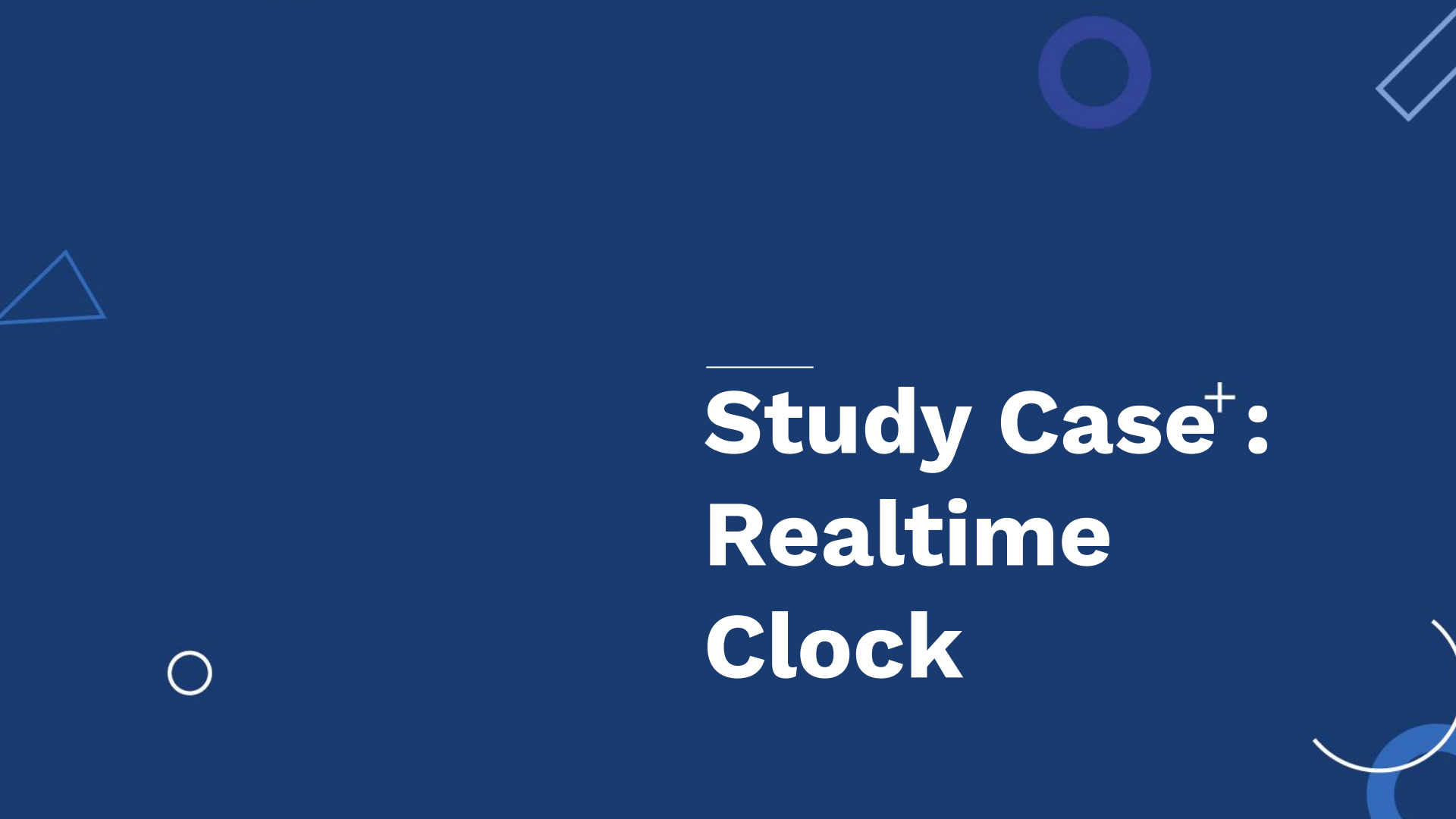
Sesi 12 | Fetch Data - Penerapan Ajax Xhr

FUNCTION COMPONENT

Kita akan memanfaatkan **useEffect()** untuk melakukan data fetching dan mengisinya kepada local state

Dari contoh di samping kita dapat melihat bahwa penggunaan function component dapat membuat kode kita menjadi lebih pendek, rapi, dan mudah untuk dilihat

```
1  import React, { useEffect, useState } from 'react'
2
3  export default function Halu() {
4    const [todos, setTodos] = useState([])
5
6    useEffect(() => {
7      fetch('https://jsonplaceholder.typicode.com/todos')
8        .then((response) => response.json())
9        .then((data) => setTodos(data.slice(0, 20)))
10   })
11
12   return (
13     <div>
14       <table className="table">
15         <tbody>
16           {
17             todos.map((todo, index) => (
18               <tr key={index}>
19                 <td>{todo.id}</td>
20                 <td>{todo.title}</td>
21                 <td>{todo.completed ? "v" : "x"}</td>
22               </tr>
23             ))
24           }
25         </tbody>
26       </table>
27     </div>
28   )
29 }
30
```



Study Case⁺: Realtime Clock

Sesi 12 | Study Case : Realtime Clock

Mari kita implementasikan materi lifecycle dengan sebuah studi kasus yang menarik. Kita akan membuat realtime clock.

Flow nya adalah sebagai berikut :

1. State akan di initiate dengan waktu sekarang
2. Kita akan buat sebuah function tick() untuk update state dengan waktu yang terbaru
3. Kita akan memanfaatkan fasilitas lifecycle untuk memulai timer

Untuk implementasi pertama, kita akan menggunakan Class Component. Silakan teman-teman initiate project baru dengan perintah :

```
> create-react-app realtime-clock
```

Setelah inisiasi selesai, mari kita masuk ke dalam folder yang terbentuk dan memulai implementasi



Sesi 12 | Study Case : Realtime Clock

CLASS COMPONENT

1. Update App.js menjadi seperti di bawah ini

```
1  import React from 'react';
2  import './App.css';
3
4  class App extends React.Component {
5    render() {
6      return (
7        <div className="App">
8          <h1>Realtime CLOCK</h1>
9          <hr />
10         </div>
11       );
12     }
13   }
14
15   export default App;
```

2. Update App.css menjadi seperti di bawah ini

```
1  .App {
2    width: 500px;
3    margin: 0 auto;
4    text-align: center;
5  }
```

3. Tambahkan State

```
5  constructor() {
6    super()
7    this.state = {
8      date: new Date()
9    }
10 }
```


Sesi 12 | Study Case : Realtime Clock

CLASS COMPONENT

4. Tambahkan function tick()

```
12 |     tick() {  
13 |         this.setState({  
14 |             date: new Date()  
15 |         })  
16 |     }
```

6. Tambahkan lifecycle componentDidMount()

```
18 |     componentDidMount() {  
19 |         this.timerID = setInterval(() => this.tick(), 1000)  
20 |     }
```

5. Tampilkan di render

```
26 |     render() {  
27 |         return (  
28 |             <div className="App">  
29 |                 <h1>Realtime CLOCK</h1>  
30 |                 <hr />  
31 |                 <h1>{this.state.date.toLocaleTimeString()}</h1>  
32 |             </div>  
33 |         );  
34 |     }
```

7. Tambahkan lifecycle componentWillUnmount()

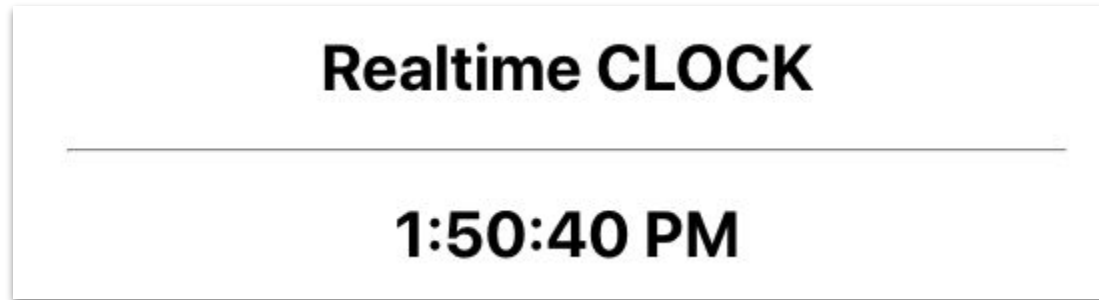
```
22 |     componentWillUnmount() {  
23 |         clearInterval(this.timerID)  
24 |     }
```



Sesi 12 | Study Case : Realtime Clock

CLASS COMPONENT

8. Jalankan aplikasi, tampilan yang diharapkan adalah sebagai berikut



Selanjutnya, mari kita implementasikan dengan functional component

Sesi 12 | Study Case : Realtime Clock

FUNCTIONAL COMPONENT

1. Update App.js menjadi seperti di bawah ini

```
1  import React, { useState } from 'react';
2  import './App.css';
3
4  function App() {
5    return (
6      <div className="App">
7        <h1>Realtime CLOCK</h1>
8        <hr />
9      </div>
10    );
11  }
12
13  export default App;
```

2. Tambahkan State

```
8  const [date, setDate] = useState(new Date())
```

3. Tambahkan function tick()

```
5  function tick() {
6    setDate(new Date())
7  }
```

4. Tambahkan setInterval

```
11  setInterval(() => tick(), 1000)
```



Sesi 12 | Study Case : Realtime Clock

FUNCTIONAL COMPONENT

5. Update function return menjadi seperti ini

```
13   return (  
14     <div className="App">  
15       <h1>Realtime CLOCK</h1>  
16       <hr />  
17       <h1>{date.toLocaleTimeString()}</h1>  
18     </div>  
19   );  
20 }
```

6. Jalankan aplikasi, tampilan seharusnya sama

Realtime CLOCK

1:50:40 PM

Baik Class Component maupun Functional Component akan menampilkan tampilan yang sama. Tapi, temen-temen bisa memilih sendiri mana yang lebih efisien dari segi kode bukan ?



HACKTIV8