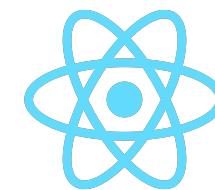


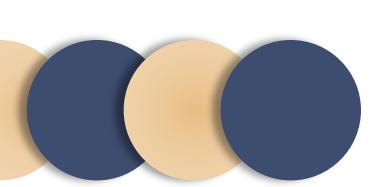


Pemrograman Sisi Cleint

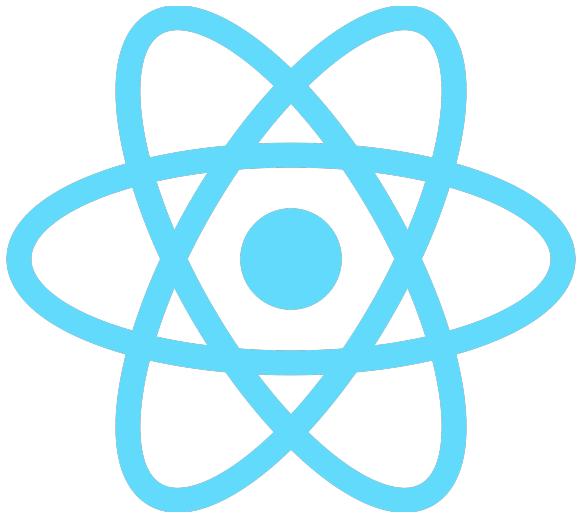
ReactJS Hooks

Muhammad Syaifur Rohman, S.Kom, M.CS





Definisi



FYI

Hooks adalah fitur yang diperkenalkan di React 16.8 yang memungkinkan penggunaan state dan fitur-fitur React lainnya tanpa menulis kelas. Hooks memungkinkan Anda untuk mengelola state, efek samping, context, dan banyak lagi dalam komponen fungsional.



Persiapan

Kapan ?

- Ketika Anda ingin menggunakan state dan fitur-fitur lain dari React dalam komponen fungsional.
- Untuk mengelola efek samping seperti pengambilan data atau pengaturan timer.
- Untuk memanfaatkan context atau mengelola state yang lebih kompleks dengan **useReducer**.



Persiapan

Why

- React Hooks adalah fitur powerful yang memungkinkan pengelolaan state dan efek samping dalam komponen fungsional dengan cara yang lebih bersih dan terstruktur. Dengan memahami dan menggunakan hooks seperti **useState**, **useEffect**, **useContext**, serta hooks tambahan seperti **useReducer**, **useCallback**, **useMemo**, dan **useRef**, Anda dapat membangun komponen React yang lebih modular dan dapat dipelihara. Custom hooks memungkinkan Anda untuk mengekstrak dan menggunakan kembali logika stateful di berbagai komponen, menjadikan kode Anda lebih DRY (Don't Repeat Yourself) dan mudah dipelihara.

Persiapan

```
src/
  └── components/
    ├── Counter.js
    ├── Timer.js
    ├── ThemedComponent.js
    ├── CounterReducer.js
    ├── CounterCallback.js
    ├── ExpensiveCalculation.js
    └── TextInput.js
  └── context/
    └── ThemeContext.js
  └── App.js
  └── index.js
```

useState

useState adalah hook yang digunakan untuk mendeklarasikan state dalam komponen fungsional.

```
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}

export default Counter;
```

TRY... 

- **useState** mengembalikan array dengan dua elemen: nilai state saat ini (**count**) dan fungsi untuk memperbarui nilai state (**setCount**).
- Anda dapat menginisialisasi state dengan nilai awal (dalam contoh ini, 0).

useEffect

useEffect adalah hook yang digunakan untuk menjalankan efek samping dalam komponen, seperti pengambilan data, pengaturan timer, atau memperbarui DOM.

```
import React, { useState, useEffect } from 'react';

function Timer() {
  const [seconds, setSeconds] = useState(0);

  useEffect(() => {
    const interval = setInterval(() => {
      setSeconds(seconds => seconds + 1);
    }, 1000);
    return () => clearInterval(interval);
  }, []);

  return <div>Seconds: {seconds}</div>;
}

export default Timer;
```

TRY...
→

- **useEffect** menerima dua argumen: fungsi yang akan dijalankan dan array dependensi.
- Efek dijalankan setelah render pertama dan setiap kali nilai dalam array dependensi berubah.
- Membersihkan efek dengan mengembalikan fungsi pembersihan dari **useEffect**.

useContext

useContext adalah hook yang digunakan untuk mengakses context dalam komponen fungsional.

useContext digunakan untuk mengakses nilai dari context (ThemeContext) yang disediakan oleh ThemeProvider.

```
import React, { createContext, useContext, useState } from 'react';

const ThemeContext = createContext();

function ThemeProvider({ children }) {
  const [theme, setTheme] = useState('light');

  const toggleTheme = () => {
    setTheme(prevTheme => (prevTheme === 'light' ? 'dark' : 'light'));
  };

  return (
    <ThemeContext.Provider value={{ theme, toggleTheme }}>
      {children}
    </ThemeContext.Provider>
  );
}

function ThemedComponent() {
  const { theme, toggleTheme } = useContext(ThemeContext);

  return (
    <div style={{ background: theme === 'light' ? '#fff' : '#333',
      color: theme === 'light' ? '#000' : '#fff' }}>
      <p>Current theme: {theme}</p>
      <button onClick={toggleTheme}>Toggle Theme</button>
    </div>
  );
}

function App() {
  return (
    <ThemeProvider>
      <ThemedComponent />
    </ThemeProvider>
  );
}

export default App;
```

TRY...
→

useReducer

useReducer adalah alternatif untuk `useState` yang cocok untuk state yang kompleks dan logika pembaruan state yang melibatkan banyak tipe aksi.

useReducer menerima reducer dan initial state, dan mengembalikan state saat ini serta fungsi dispatch.

```
import React, { useReducer } from 'react';

const initialState = { count: 0 };

function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return { count: state.count + 1 };
    case 'decrement':
      return { count: state.count - 1 };
    default:
      throw new Error();
  }
}

function Counter() {
  const [state, dispatch] = useReducer(reducer, initialState);

  return (
    <div>
      <p>Count: {state.count}</p>
      <button onClick={() => dispatch({ type: 'increment' })}>Increment</button>
      <button onClick={() => dispatch({ type: 'decrement' })}>Decrement</button>
    </div>
  );
}

export default Counter;
```

TRY...
→

useCallback

useCallback mengembalikan versi memoized dari fungsi callback yang hanya berubah jika salah satu dependensinya berubah. Ini berguna untuk mengoptimalkan kinerja pada komponen yang bergantung pada fungsi callback.

useCallback digunakan untuk menghindari pembuatan ulang fungsi increment pada setiap render.

```
import React, { useState, useCallback } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  const increment = useCallback(() => {
    setCount(count + 1);
  }, [count]);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
    </div>
  );
}

export default Counter;
```

TRY...
→

useMemo

useMemo mengembalikan nilai memoized yang hanya dihitung ulang jika salah satu dependensinya berubah. Ini berguna untuk mengoptimalkan kinerja dengan menghindari perhitungan ulang yang mahal.

useMemo digunakan untuk menghindari perhitungan ulang result kecuali jika count berubah.

```
import React, { useState, useMemo } from 'react';

function ExpensiveCalculation({ count }) {
  const result = useMemo(() => {
    // Simulasi perhitungan yang mahal
    let sum = 0;
    for (let i = 0; i < 1000000000; i++) {
      sum += i;
    }
    return sum + count;
  }, [count]);
  return <div>Result: {result}</div>;
}

function App() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <button onClick={() => setCount(count + 1)}>Increment</button>
      <ExpensiveCalculation count={count} />
    </div>
  );
}

export default App;
```

TRY...
→

useRef

useRef digunakan untuk mengakses elemen DOM langsung atau menyimpan nilai yang tidak memicu render ulang ketika berubah.

useRef digunakan untuk menyimpan referensi ke elemen input dan memanggil metode fokus pada elemen tersebut.

```
import React, { useRef } from 'react';

function TextInput() {
  const inputRef = useRef(null);

  const focusInput = () => {
    inputRef.current.focus();
  };

  return (
    <div>
      <input ref={inputRef} type="text" />
      <button onClick={focusInput}>Focus Input</button>
    </div>
  );
}

export default TextInput;
```

TRY...
→

Custom Hooks

Custom hooks memungkinkan Anda untuk mengekstrak logika stateful ke dalam fungsi yang dapat digunakan kembali.

useFetch adalah custom hook yang mengelola pengambilan data dari URL yang diberikan dan mengembalikan data dan status loading.

```
import React, { useState, useEffect } from 'react';

function useFetch(url) {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetch(url)
      .then(response => response.json())
      .then(data => {
        setData(data);
        setLoading(false);
      });
  }, [url]);

  return { data, loading };
}

function DataFetchingComponent() {
  const { data, loading } =
    useFetch('https://jsonplaceholder.typicode.com/todos/1');

  if (loading) {
    return <p>Loading...</p>;
  }

  return <div>{data.title}</div>;
}

export default DataFetchingComponent;
```

TRY...
→

Hasil

Count: 0

Increment

Seconds: 2

test ini

Focus Input

Current theme: dark

Toggle Theme

Count: -4

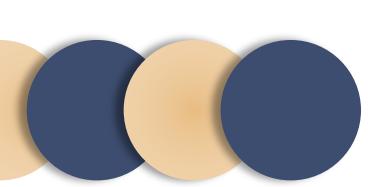
Increment Decrement

Result: 499999999067109000

Count: 4

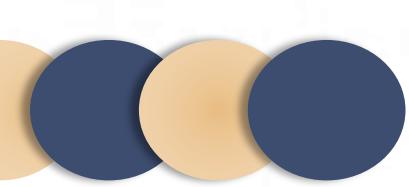
Increment

delectus aut autem



Best Practice

- React Hooks adalah fitur powerful yang memungkinkan pengelolaan state dan efek samping dalam komponen fungsional dengan cara yang lebih bersih dan terstruktur. Dengan memahami dan menggunakan hooks seperti **useState**, **useEffect**, **useContext**, serta hooks tambahan seperti **useReducer**, **useCallback**, **useMemo**, dan **useRef**, Anda dapat membangun komponen React yang lebih modular dan dapat dipelihara. Custom hooks memungkinkan Anda untuk mengekstrak dan menggunakan kembali logika stateful di berbagai komponen, menjadikan kode Anda lebih DRY (Don't Repeat Yourself) dan mudah dipelihara.



Referensi

1. Eran Krinsbruner, **A Frontend Web Developer's Guide to Testing**, 2022
2. Adam Boduch , Roy Derks , Mikhail Sakhniuk, **React and React Native - Fourth Edition**, 2022
3. Maya Shavin , Raymond Camden, **Frontend Development Projects with Vue.js 3 - Second Edition**, 2023
4. Waweru Mwaura, **End-to-End Web Testing with Cypress**, 2021
5. <https://roadmap.sh/frontend>
6. <https://jsonplaceholder.typicode.com/users>



Thanks

Muhammad Syaifur Rohman, S.Kom, M.CS

